Deep Domain Adaptation With Max-Margin Principle for Cross-Project Imbalanced Software Vulnerability Detection

VAN NGUYEN, Monash University, Australia
TRUNG LE, Monash University, Australia
CHAKKRIT TANTITHAMTHAVORN, Monash University, Australia
JOHN GRUNDY, Monash University, Australia
DINH PHUNG, Monash University, Australia

Software vulnerabilities (SVs) have become a common, serious, and crucial concern due to the ubiquity of computer software. Many AI-based approaches have been proposed to solve the software vulnerability detection (SVD) problem to ensure the security and integrity of software applications (in both the development and testing phases). However, there are still two open and significant issues for SVD in terms of i) learning automatic representations to improve the predictive performance of SVD, and ii) tackling the scarcity of labeled vulnerability datasets that conventionally need laborious labeling effort by experts. In this paper, we propose a novel approach to tackle these two crucial issues. We first exploit the automatic representation learning with deep domain adaptation for SVD. We then propose a novel cross-domain kernel classifier leveraging the max-margin principle to significantly improve the transfer learning process of SVs from imbalanced labeled into imbalanced unlabeled projects. *Our approach is the first work that leverages solid body theories of the max-margin principle, kernel methods, and bridging the gap between source and target domains for imbalanced domain adaptation (DA) applied in cross-project SVD.* The experimental results on real-world software datasets show the superiority of our proposed method over state-of-the-art baselines. In short, our method obtains a higher performance on F1-measure, one of the most important measures in SVD, from 1.83% to 6.25% compared to the second highest method in the used datasets.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Security and privacy**;

Additional Key Words and Phrases: Software Security, Automated Cross-Project Vulnerability Detection

ACM Reference Format:
Van Nguyen, Trung Le, Chakkrit Tantithamthavorn, John Grundy, and Dinh Phung. 2024. Deep Domain Adaptation With Max-Margin Principle for Cross-Project Imbalanced Software Vulnerability Detection. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (April 2024), 34 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Software vulnerabilities (SVs), defined as specific flaws or oversights in software programs allowing attackers to exploit the code base and potentially undertake dangerous activities (e.g., exposing or altering sensitive information, disrupting, degrading or destroying a system, or taking control of a program or computer system) [Dowd et al.(2006), Fu et al.(2024b)], are very common and represent major security risks due to the ubiquity of computer software. Detecting and eliminating software

Authors' addresses: Van Nguyen, Monash University, Clayton, Australia, van.nguyen1@monash.edu; Trung Le, Monash University, Clayton, Australia, trunglm@monash.edu; Chakkrit Tantithamthavorn, Monash University, Clayton, Australia, chakkrit@monash.edu; John Grundy, Monash University, Clayton, Australia, john.grundy@monash.edu; Dinh Phung, Monash University, Clayton, Australia, dinh.phung@monash.edu.

1049-331X/2024/4-ART $15.00
https://doi.org/XXXXXXX.XXXXXXX

vulnerabilities are hard as software development technologies and methodologies vary significantly between projects and products [Fu et al.(2022), Liu et al.(2022), Thongtanunam et al.(2022), Fu et al.(2023a), Fu et al.(2024a)]. The severity of the threat imposed by software vulnerabilities (SVs) has significantly increased over the years causing significant damages to companies and individuals. The worsening software vulnerability situation has necessitated the development of automated advanced approaches and tools that can efficiently and effectively detect SVs with a minimal level of human intervention.

Software vulnerability detection (SVD) is crucial in software engineering to ensure the security and integrity of software applications [Dowd et al.(2006), Lin et al.(2020), Hanif et al.(2021), Nguyen et al.(2021), Fu and Tantithamthavorn(2022), Liu et al.(2023), Fu et al.(2023b)]. The ability to identify vulnerable programs or functions is a critical part of the security engineering process which helps security professionals efficiently allocate their resources and address severe vulnerabilities (in the development and testing phases), ultimately enhancing the security and reliability of the software application. To respond to this demand, many vulnerability detection systems and methods, ranging from open source to commercial tools, and from manual to automatic methods [Neuhaus et al.(2007), Shin et al.(2011), Grieco et al.(2016), Li et al.(2018b), Duan et al.(2019), Cheng et al.(2019), Wattanakriengkrai et al.(2020), Pornprasit and Tantithamthavorn(2021)] have been proposed and implemented.

Most previous work in software vulnerability detection (SVD), such as [Yamaguchi et al.(2011), Shin et al.(2011), Li et al.(2016), Grieco et al.(2016), Kim et al.(2017)], has based primarily on handcrafted features which are manually chosen by knowledgeable domain experts with possibly outdated experience and underlying biases. In many situations, these handcrafted features do not generalize well. For example, features that work well in a certain software project may not perform well in other projects [Zimmermann et al.(2009)]. To alleviate the dependency on handcrafted features, the use of automatic features in SVD, leveraging deep learning (DL) techniques, has been studied [Li et al.(2018b), Dam et al.(2018), Li et al.(2018a), Pornprasit and Tantithamthavorn(2022), Nguyen et al.(2022a), Fu et al.(2024c)]. These DL-based approaches show the advantages of employing automatic features over handcrafted features for addressing the SVD problem.

Another major challenging issue in SVD is the scarcity of labeled software projects that are needed in order to train the machine learning and deep learning SVD models. The process of labeling vulnerable source code is tedious, time-consuming, error-prone, and can be very challenging even for domain experts. This has resulted in few labeled projects compared with a vast volume of unlabeled ones. Some recent approaches [Nguyen et al.(2019), Nguyen et al.(2020), Liu et al.(2020)] have been proposed to solve this challenging problem with the aim to transfer the learning of vulnerabilities from labeled source domains to unlabeled target domains. Particularly, the methods in [Nguyen et al.(2019), Nguyen et al.(2020)] learn domain-invariant features from the source code data of the source and target domains by using the adversarial learning framework such as generative adversarial network (GAN) [Goodfellow et al.(2014)] while the method in [Liu et al.(2020)] consists of many subsequent stages: i) pre-training a deep feature model for learning representation of token sequences (i.e., source code data), ii) learning cross-domain representations using a transformation to project token sequence embeddings from (i) to a latent space, and iii) training a classifier from the representations of the source domain data obtained from (ii). *However, none of these methods exploit the imbalanced nature of source code projects for which the vulnerable data points are significantly minor compared to non-vulnerable ones.* Without a robust capability to learn from small amounts of data (vulnerable data), SVD models are likely to be more influenced by large amounts of data (non-vulnerable). This can potentially diminish their ability to effectively detect vulnerabilities in cross-domain vulnerability classification. The negative effect of this issue can be exhibited via the F1-measure of the models when applied to the target domain.

On the other hand, kernel methods with the max-margin principle are widely recognized for their effectiveness in handling imbalanced datasets [Schölkopf et al.(2001), Kotsiantis et al.(2006), Tsang et al.(2007), Le et al.(2010), Le et al.(2014), Duong et al.(2015)]. In addition, they can help achieve high generalization performance even with a relatively small number of data points. To simplify, they create a defined region in the feature space, often represented as a simple geometric shape such as a half-hyperplane [Schölkopf et al.(2001)] or hypersphere [Tax and Duin(2004), Tsang et al.(2005), Tsang et al.(2007)], to encapsulate the majority class, which typically corresponds to non-vulnerable data in the context of SVD. The key idea is that a simple domain of majority in the feature space, when being mapped back to the input space, forms a set of contours that can distinguish majority data from minority data (i.e., vulnerable data).

In this paper, by leveraging learning domain-invariant features and kernel methods with the max-margin principle, we propose ***D**omain **A**daptation with **M**ax-**M**argin **P**rinciple* (DAM2P) to efficiently transfer the learning of vulnerabilities from imbalanced labeled source domains to imbalanced unlabeled target domains. Inspired by the max-margin principle proven efficiently and effectively for learning from imbalanced data, when learning domain-invariant features between the source and target domains, we propose to learn a max-margin hyperplane (i.e., cross-domain kernel classifier) on the feature space to separate vulnerable and non-vulnerable data.

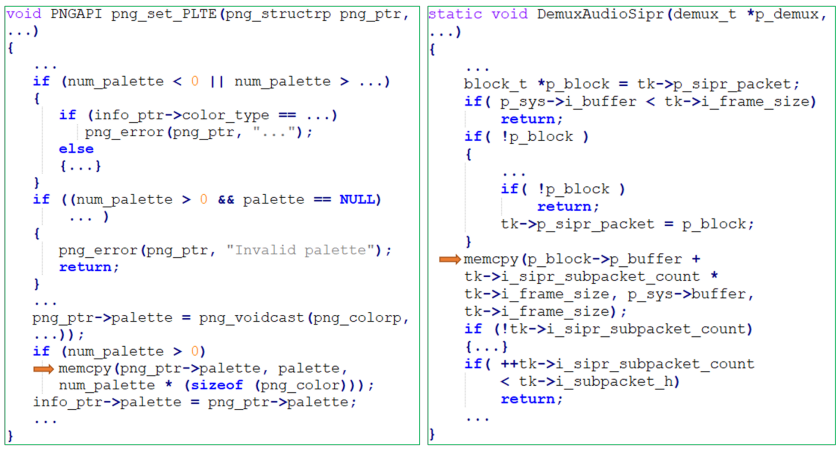
More specifically, we combine labeled source domain data and unlabeled target domain data and then learn a hyperplane to separate *labeled source domain non-vulnerable from vulnerable data* and *unlabeled target domain data from the origin* such that the margin is maximized. In addition, the margin is defined as the minimization of the source domain and target domain margins in which the *source domain margin* is regarded as the minimal distance from vulnerable data points to the hyperplane, while the *target domain margin* is regarded as the distance from the origin (i.e., *the (0,0) coordinate in the feature space*) to the hyperplane [Schölkopf et al.(2001)]. Furthermore, it is worth noting that in our proposed cross-domain kernel classifier, the source domain margin is utilized to leverage information from the labeled source domain in constructing the classifier within the feature space. Meanwhile, the target domain margin plays a crucial role in harnessing information from the unlabeled target domain, contributing to the ongoing process of updating and enhancing the classifier's capability in vulnerability transfer learning.

Our key contributions in this work include:

- We propose a novel approach named DAM2P for the topical problem of cross-domain imbalanced SVD. In particular, in our proposed method, we leverage learning domain-invariant features and kernel methods with the max-margin principle that can bridge the gap between the source and target domains on a joint space while being able to tackle efficiently and effectively the imbalanced nature of the source and target domains to significantly improve the transfer learning process of SVs from imbalanced labeled projects into imbalanced unlabeled ones. *To the best of our knowledge, our approach is the first work that leverages solid body theories of the max-margin principle, kernel methods, and bridging the gap between source and target domains for imbalanced domain adaptation (DA) applied in cross-project SVD.*
- We conduct extensive experiments on five real-world software datasets consisting of FFmpeg, LibTIFF, LibPNG, VLC, and Pidgin software projects. It is worth noting that to demonstrate and compare the capability of our proposed method and baselines in the transfer learning for SVD, the datasets (FFmpeg, VLC, and Pidgin) from the *multimedia application domain* are used as the source domains whilst the datasets (LibPNG and LibTIFF) from the *image application domain* are used as the target domains. The experimental results show that our method significantly outperforms the baselines by a wide margin, especially for the F1-measure, one of the most important measures in SVD [Li et al.(2016), Li et al.(2018a), Nguyen et al.(2019)].

2 MOTIVATING EXAMPLE

Figure 1 shows an example of source code functions obtained from the open-source VLC and LibPNG projects. Both functions from the VLC and LibPNG projects depicted in Figure 1 invoke the *memcpy* function which is used to copy the contents of one buffer to another buffer. The misuse of this function can cause a buffer overflow if there is insufficient memory allocated in the target buffer for all of the contents to be copied from the source buffer.



```
void PNGAPI png_set_PLTE(png_structrp png_ptr,
...)
{
   ...
   if (num_palette < 0 || num_palette > ...)
   {
      if (info_ptr->color_type == ...)
         png_error(png_ptr, "...");
      else
      {...}
   }
   if ((num_palette > 0 && palette == NULL)
       ... )
   {
      png_error(png_ptr, "Invalid palette");
      return;
   }
   ...
   png_ptr->palette = png_voidcast(png_colorp,
   ...));
   if (num_palette > 0)
   memcpy(png_ptr->palette, palette,
      num_palette * (sizeof (png_color)));
   info_ptr->palette = png_ptr->palette;
   ...
}
```

```
static void DemuxAudioSipr(demux_t *p_demux,
...)
{
    ...
    block_t *p_block = tk->p_sipr_packet;
    if( p_sys->i_buffer < tk->i_frame_size)
        return;
    if( !p_block )
    {
        ...
        if( !p_block )
            return;
        tk->p_sipr_packet = p_block;
    }
    memcpy(p_block->p_buffer +
    tk->i_sipr_subpacket_count *
    tk->i_frame_size, p_sys->buffer,
    tk->i_frame_size);
    if (!tk->i_sipr_subpacket_count)
    {...}
    if( ++tk->i_sipr_subpacket_count
        < tk->i_subpacket_h)
        return;
    ...
}
```

Fig. 1. An example of two C/C++ source code functions obtained from the LibPNG project (Left) and VLC project (Right). These two source code examples highlight the same buffer overflow vulnerability due to the misuse of the *memcpy* function.

To demonstrate that transfer learning for software vulnerability detection between source code data (e.g., functions) from different projects (i.e., domains) is plausible and promising, we observe that the functions (from different projects) in Figure 1 are written in different ways, but share *similar semantic relationships (i.e., the similar nature types of vulnerabilities)*. Therefore, a model that can capture the characteristics of the first function in the first project should ideally be able to accurately predict the second function in the second project. Thus it makes sense to undertake transfer learning from the first project to the second project.

We find many other source code functions across the projects that suffer from similar vulnerability types. This observation provides us with the motivation and incentive to undertake transfer learning from a labeled software project to another unlabeled software project. We present our novel approach – Domain Adaptation with Max-Margin Principle (DAM2P) – proposed for solving cross-project imbalanced software vulnerability detection. Our DAM2P approach allows us to transfer a deep vulnerability classifier obtained from an imbalanced labeled software project to another imbalanced unlabeled software project. Compared to the state-of-the-art baselines, our approach is one of the very first methods taking into account the imbalanced nature of source code data, for which the vulnerable data points are significantly minor compared to non-vulnerable ones, in cross-domain SVD for successfully boosting the vulnerability transfer learning process.

Our goal in tackling cross-domain software vulnerability detection is twofold. Firstly, we aim to address the scarcity of labeled source code projects required for training machine learning and deep learning models to identify vulnerabilities in source code data. Secondly, we seek to directly aid software engineers in identifying vulnerable programs or functions in source code projects from various domains. This assistance is provided through the application of a vulnerability classifier learned from a specific domain project.

In the scope of our paper, in our experiments, we consider different software domains are those that come from different software applications (e.g., multimedia applications and image applications) written in the same C/C++ programming language. These domains are written in different ways (e.g., using different structures or variable names); however, they share the same nature (types) of code vulnerabilities. This property lets our proposed method and the baselines be deployed realistically for cross-domain software vulnerability detection.

3 RELATED WORK

Automatic features in SVD have been studied [Li et al.(2018b), Lin et al.(2018), Dam et al.(2018), Li et al.(2018a), Duan et al.(2019), Cheng et al.(2019), Zhuang et al.(2020)] due to its advantages of employing automatic features over handcrafted features. In particular, [Dam et al.(2018)] employed a deep neural network to transform sequences of code tokens to vectorial features that are further fed to a separate classifier; whereas [Li et al.(2018b)] combined the learning of the vector representation and the training of the classifier in a deep network. Advanced deep net architectures have been investigated for the SVD problem. [Russell et al.(2018)] combined both recurrent neural networks (RNNs) and convolutional neural networks (CNNs) for feature extraction from the embedded source code representations while [Zhuang et al.(2020)] proposed a model for smart contract vulnerability detection based on a graph neural network [Kipf and Welling(2016)].

Deep DA-based methods have been recently studied for cross-domain (cross-project) SVD. Notably, [Nguyen et al.(2019)] proposed a novel architecture and employed the adversarial learning framework (e.g., GAN) to learn domain-invariant features that can be transferred from labeled source to unlabeled target code projects. [Nguyen et al.(2020)] enhanced [Nguyen et al.(2019)] by proposing an elegant workaround to combat the mode collapsing problem possibly faced in that work due to the use of GAN. Finally, [Liu et al.(2020)] proposed a multi-stage approach with three sequential stages: i) pre-training a deep model for learning the representation of token sequences (i.e., source code data), ii) learning cross-domain representations using a transformation to project token sequence embeddings from (i) to a latent space, iii) training a classifier from the representations of the source domain data obtained from (ii). The trained classifier is then applied to the target domain.

In computer vision, DA has been intensively studied and showed appealing performance in various transfer learning tasks, notably DDAN [Ganin and Lempitsky(2015)], MMD [Long et al.(2015)], D2GAN [Nguyen et al.(2017)], DIRT-T [Shu et al.(2018)], HoMM [Chen et al.(2020)], and LAMDA [Le et al.(2021)]. Most of the introduced methods were claimed, applied, and showed the results for vision data. There is no evidence that they can straightforwardly be applied to source code data in cross-project SVD. *It is worth noting that cross-project SVD is special because the source code data are more complicated and different from text and image data due to consisting of complex semantic and syntactic relationships between statements and tokens.* In our paper, inspired by [Nguyen et al.(2019)], we borrowed the principles of some well-known and state-of-the-art methods, e.g., DDAN, MMD, D2GAN, DIRT-T, HoMM, and LAMDA, and refactored them using the CDAN architecture introduced in [Nguyen et al.(2019)] for cross-project SVD to compare with our proposed approach.

We note that our method is different from previous baselines, which also use deep domain adaptation (DA) for cross-project SVD such as [Nguyen et al.(2019), Nguyen et al.(2020), Liu et al.(2020)]*, in proposing a novel cross-domain kernel classifier leveraging max-margin kernel methods for handling imbalanced nature of the source code data. To the best of our knowledge, none of the previous cross-project SVD approaches exploit and address the imbalanced nature existing in source code projects for cross-project SVD.*

4 RELATED BACKGROUND

4.1 Kernel methods

Kernel methods are a class of machine learning techniques widely recognized for their versatility and effectiveness in various domains. At their core, they transform data into a higher-dimensional space, allowing for the discovery of complex patterns and relationships that might not be readily apparent in the original data space. This transformation is achieved using mathematical kernel functions (e.g., *Linear kernel, Polynomial kernel, or Gaussian kernel* taking input vectors in the original space and returning the dot product of the vectors in the feature space [Hearst et al.(1998), Schölkopf and Smola(2002), Hofmann et al.(2008), Nguyen et al.(2014), Le et al.(2015), Le et al.(2016)]), which quantify the similarity or dissimilarity between data points.

One of the key strengths of kernel methods is their ability to handle both linear and non-linear relationships, making them particularly useful in tasks such as classification, regression, and clustering. They excel in scenarios with imbalanced datasets, enabling the identification of rare events or minority classes. In addition, they can help achieve high generalization performance even with a relatively small number of data points [Schölkopf et al.(2001), Hsu and Lin(2002), Kotsiantis et al.(2006), Tsang et al.(2007), Le et al.(2010)]. Some popular examples of kernel methods include Support Vector Machines (SVMs) [Hearst et al.(1998)] and Kernel Principal Component Analysis (Kernel PCA) [Weston et al.(2003)]. These methods find applications in various fields, including image analysis, natural language processing, and bioinformatics, making them valuable tools for extracting meaningful insights from complex data.

Support vector machines. The support vector machine (SVM) was first introduced in the early 1960s to create a linear decision boundary in the input space [Vapnik and Lerner(1963)]. However, between 1992 and 1995, it was extended to establish a linear decision boundary in the feature space while allowing for non-linear decision boundaries in the input space, as detailed in the works of [Cortes and Vapnik(1995)]. Since then, SVM has evolved into a leading classifier with various adaptations [Schölkopf et al.(2000), Lin and Wang(2002)].

SVM has emerged as one of the most frequently utilized methods for addressing pattern recognition challenges. Its exceptional generalization capabilities, along with its capacity to learn from any dataset with minimal error, have made it widely applicable in real-world scenarios. The core concept of SVM involves mapping data from the input space to the feature space and learning an optimal hyperplane in such a way that the margin, representing the distance between the closest training set vector and the hyperplane, is maximized. This emphasis on maximizing the margin is what drives the learning capacity of the hyperplane.

Let the training set be $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_l, y_l)\}$ where $y_1, y_2, ..., y_l \in \{-1, 1\}$ are labels. Let ϕ be the transformation from the input space to the feature space. Driven by the structural risk minimization principle, for minimizing the empirical risk and maximizing the generalization capacity, the SVM optimization problem was introduced as follows:

$$\min_{\mathbf{w},b}(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i)$$

subject to

$$y_i(\mathbf{w}^\top\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i,\ i = 1, ..., l$$

where $h(\boldsymbol{x}) = \mathbf{w}^\top\phi(\boldsymbol{x}) + b$ is an optimal hyperplane while $\xi_i \geq 0$ with $i = 1, ..., l$ are slack variables, and C is a positive constant regarded as a trade-off parameter. With the slack variables, SVMs can be applicable to the case where the data are linearly inseparable.

5 DOMAIN ADAPTATION WITH MAX-MARGIN PRINCIPLE (DAM2P)

In the following sections, we elucidate the workings of our proposed DAM2P method and its approach for addressing the cross-domain imbalanced software vulnerability detection problem. First, we articulate the problem statement associated with cross-domain imbalanced software vulnerability detection. We then expound on the utilization of deep domain adaptation, employing an adversarial learning framework such as GAN, to learn domain-invariant features. Finally, we illustrate how our proposed cross-domain kernel classifier, leveraging the max-margin principle, operates. It integrates with the process of learning domain-invariant features to not only bridge the gap between the source and target domains in a joint space but also to efficiently and effectively handle the imbalanced nature of both source and target domains to significantly enhance the transfer learning process of software vulnerabilities from imbalanced labeled projects to imbalanced unlabeled other projects.

5.1 Problem Statement

Given a labeled source domain dataset $S = \left\{\left(\boldsymbol{x}_1^S, y_1\right), \ldots, \left(\boldsymbol{x}_{N_S}^S, y_{N_S}\right)\right\}$ where $y_i \in \{-1, 1\}$ (i.e., 1: vulnerable code and -1: non-vulnerable code), let $\boldsymbol{x}_i^S = \left[\boldsymbol{x}_{i1}^S, \ldots, \boldsymbol{x}_{iL}^S\right]$ be a code function represented as a sequence of L embedding vectors. We note that each embedding vector corresponds to a statement in the code function. Similarly, the unlabeled target domain dataset $T = \left\{\boldsymbol{x}_1^T, \ldots, \boldsymbol{x}_{N_T}^T\right\}$ consists of many code functions where each code function $\boldsymbol{x}_i^T = \left[\boldsymbol{x}_{i1}^T, \ldots, \boldsymbol{x}_{iL}^T\right]$ is a sequence of L embedding vectors.

In standard DA approaches, domain-invariant features are learned on a joint space so that a classifier mainly trained based on labeled source domain data can be transferred to predict well unlabeled target domain data. The classifiers of interest are usually deep nets conducted on top of domain-invariant features. In this work, by leveraging the kernel theory and the max-margin principle, we consider a kernel machine on top of domain-invariant features, which is a hyperplane on a feature space via a feature map ϕ.

Inspired by the max-margin principle proven efficiency and effectiveness for learning from imbalanced data, when learning domain-invariant features, we propose to learn a max-margin hyperplane on the feature space to separate vulnerable (small amounts of code) and non-vulnerable (large amounts of code) data. More specifically, we combine labeled source domain data and unlabeled target domain data, and then learn a hyperplane to separate *source domain non-vulnerable from vulnerable data* and *target domain data from the origin* such that the margin is maximized. Here, the margin is defined as the minimization of the source domain and target domain margins in which the *source domain margin* is defined as the minimum distance from vulnerable data points to hyperplane while the *target domain margin* is defined as the distance from the origin (i.e., the (0,0) coordinate in the feature space) to the hyperplane [Schölkopf et al.(2001)].

5.2 Our Proposed Approach

We aim to build a model that can effectively be used for cross-domain vulnerability detection where the vulnerability detection classifier learned from a labeled source domain can be transferred to classify the data from an unlabeled target domain. That directly aids software engineers in identifying vulnerable programs or functions in source code projects from various domains.

It is essential to emphasize the presence of a data representation gap between the source and target domains in cross-domain vulnerability detection [Nguyen et al.(2019), Nguyen et al.(2020)]. Hence, to ensure the effective application of the classifier trained on the source domain for classifying data from the target domain, the initial component of our proposed method focuses on bridging the gap

between these domains. To do that, we leverage the power of deep domain adaptation using the adversarial training principle, an effective and popular approach, for learning domain-invariant features as presented below.

5.2.1 ***Deep Domain Adaptation for Learning Domain-Invariant Features.*** In what follows, we present the architecture of our generator G and how to use an adversarial learning framework, GAN [Goodfellow et al.(2014)], to learn domain-invariant features in a joint latent space specified by G. To automatically learn the key features of the sequential source code data, inspired by [Li et al.(2018b), Nguyen et al.(2019), Nguyen et al.(2020)], we apply a bidirectional recurrent neural network (bidirectional RNN) to both the source and target domains. Given a source code function $\boldsymbol{x}$ in the source domain or the target domain, we denote the output of the bidirectional RNN by $\mathcal{B}(\boldsymbol{x})$. We then use some fully connected layers to connect the output layer of the bidirectional RNN with the joint feature layer wherein we bridge the gap between the source and target domains. The generator is consequently the composition of the bidirectional RNN and the fully connected layers: $G(\boldsymbol{x}) = f(\mathcal{B}(\boldsymbol{x}))$ where $f(\cdot)$ represents the map formed by the fully connected layers.

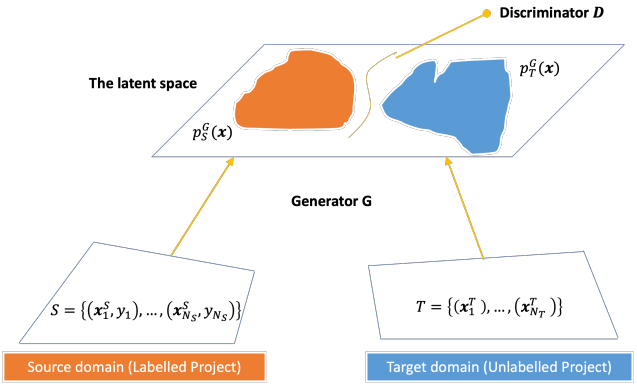


Fig. 2. A visualization of deep domain adaptation using an adversarial learning framework (i.e., GAN [Goodfellow et al.(2014)]) for learning domain-invariant features. The generator G takes the sequence of code statements (i.e., each code statement is in the vectorial form obtained from the data processing and embedding step) and maps this sequence to the joint layer (i.e., the joint space). Inspired by the GAN principle, the discriminator D is invoked to discriminate the source and target domain data while the generator G is trained to fool the discriminator D by making the source and target domain data indistinguishable. At the Nash equilibrium point, the source and target distributions are identical in the joint space.

Subsequently, to bridge the gap between the source and target domains in the latent space, inspired by GAN [Goodfellow et al.(2014)], we apply an adversarial training process. As demonstrated in Figure 2, we use a domain discriminator D to discriminate the source domain and target domain data and train the generator G to fool the discriminator D by making the source domain and target domain data indistinguishable in the latent space. The objective function is hence as follows:

$$\mathcal{H}(G, D) := \frac{1}{N_S}\sum_{i=1}^{N_S}\log D\left(G\left(\boldsymbol{x}_i^S\right)\right) + \frac{1}{N_T}\sum_{i=1}^{N_T}\log\left[1 - D\left(G\left(\boldsymbol{x}_i^T\right)\right)\right] \tag{1}$$

where we seek the optimal generator G^* and the domain discriminator D^* by solving:

$$G^* = \underset{G}{\operatorname{argmin}}\,\mathcal{H}(G, D) \text{ and } D^* = \underset{D}{\operatorname{argmax}}\,\mathcal{H}(G, D)$$

It is crucial to highlight that optimizing Eq. (1) involves training the discriminator D to maximize the probability of correctly assigning domain labels to the representations of both source and target domain data generated by the generator G. Considering a single probability scalar output by the discriminator D, updating the discriminator D by maximizing Eq. (1) requires aiming for a value close to 1 for $D\left(G\left(\boldsymbol{x}_i^S\right)\right)$ and a value close to 0 for $D\left(G\left(\boldsymbol{x}_i^T\right)\right)$. Conversely, minimizing Eq. (1) during the generator G updating process aims to achieve a value close to 1 for $D\left(G\left(\boldsymbol{x}_i^T\right)\right)$. In essence, this adversarial optimization process involves the discriminator D learning to differentiate between the representations of source and target domain data, while the generator G endeavors to make the representations of target domain data indistinguishable from those of the source domain data. Ultimately, by the end of the training process, through the generator G, the representations of source and target domain data are bridged in the latent space.

5.2.2 ***Cross-domain Kernel Classifier.*** Integrating with the process of learning domain-invariant features to bridge the gap between the source and target domains in the latent space, thereby facilitating the vulnerability transfer learning process, the second part of our proposed model involves constructing a cross-domain classifier. The classifier is designed not only to efficiently and effectively handle the imbalanced nature of both source and target domains but also to leverage the information from the unlabeled target domain to further update and enhance the classifier's capability in vulnerability transfer learning. In what follows, we illustrate how our proposed cross-domain kernel classifier, leveraging the max-margin principle (i.e., effectiveness in handling imbalanced datasets [Schölkopf et al.(2001), Hsu and Lin(2002), Kotsiantis et al.(2006), Tsang et al.(2007), Le et al.(2010)]), operates.

To build up an efficient domain adaptation approach for source code data that can tackle well the imbalanced nature of source code projects, we leverage learning domain-invariant features with the max-margin principle in the context of kernel machines to propose a novel cross-domain kernel classifier named DAM2P. In particular, inspired by the max-margin principle, we construct a hyperplane on the feature space: $\mathbf{w}^T\phi(G(\boldsymbol{x})) - \rho = 0$ with the feature map ϕ and learn this hyperplane using the max-margin principle. More specifically, we combine labeled source domain and unlabeled target domain data and then learn a hyperplane to separate *source domain non-vulnerable from source domain vulnerable data* and *target domain data from the origin* in such a way that the margin is maximized.

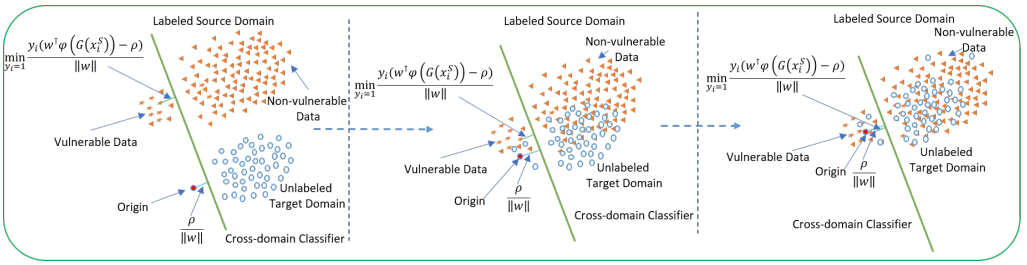


Fig. 3. The architecture of our cross-domain kernel classifier in the feature space. By using our DAM2P method, we can gradually bridge the gap between the imbalanced labeled source and imbalanced unlabeled target domains in the latent space, while in the feature space, our cross-domain kernel classifier helps to distinguish the vulnerable and non-vulnerable data. In the end, when the source and target domains are intermingled, we can transfer our trained cross-domain classifier to classify the data of the target domain.

It is worth noting that in our work, the margin is defined as the minimization of the *source* and *target margins* in which the *source margin* is the minimal distance from the source domain vulnerable data points to the hyperplane, while the *target margin* is the distance from the origin

to the hyperplane [Schölkopf et al.(2001)]. By using the source and target domain margins, our proposed cross-domain kernel vulnerability classifier can harness the information from both the source and target domain datasets in forming the optimal decision.

The overall architecture of our proposed cross-domain kernel classifier in the feature space is depicted in Figure 3 while its optimization problem is presented in Eq. (2).

Given the source domain dataset $S = \{(\boldsymbol{x}_1^S, y_1), \ldots, (\boldsymbol{x}_{N_S}^S, y_{N_S})\}$ where $y_i = 1,\ i = 1, ..., m$ and $y_i = -1,\ i = m+1, ..., N_S$ and the target domain dataset $T = \{\boldsymbol{x}_1^T, \ldots, \boldsymbol{x}_{N_T}^T\}$, we formulate the following optimization problem:

$$\max_{\mathbf{w},\rho}(\min\{\underbrace{\min_{y_i=1}\{\frac{y_i(\mathbf{w}^\top\phi(G(\boldsymbol{x}_i^S)) - \rho)}{\|\mathbf{w}\|}\}}_{\text{source margin}}, \underbrace{\frac{\rho}{\|\mathbf{w}\|}}_{\text{target margin}}\}) \tag{2}$$

subject to

$$y_i(\mathbf{w}^\top\phi(G(\boldsymbol{x}_i^S)) - \rho) \geq 0,\ i = 1, ..., N_S$$
$$\mathbf{w}^\top\phi(G(\boldsymbol{x}_i^T)) \geq \rho, i = 1, ..., N_T.$$

where $\mathbf{w}$ and ρ are the normal vector and the bias of the hyperplane and ϕ is a transformation from the joint latent space to the feature space, while G is the generator used to map the data of the source and target domains from the input space into the joint latent space.

Noting that in Eq. (2), we combine both labeled source domain and unlabeled target domain data to learn the hyperplane $(\mathbf{w}^T\phi(G(\boldsymbol{x})) - \rho = 0)$. As depicted in Figure 3, this hyperplane aims to separate (i) source domain non-vulnerable data from source domain vulnerable data and (ii) target domain data from the origin by maximizing the margin (defined as the minimization of the source and target margins). By optimizing the objective problem in Eq. (2) with the corresponding constraints, we obtain the optimal values for the parameters of the hyperplane, including $\mathbf{w}$ and ρ.

It occurs that the margin is invariant if we scale $\mathbf{w}, \rho$ by a factor $k > 0$. Hence without loosing of generality, we can assume that $\min\{\min_{y_i=1}\{y_i(\mathbf{w}^\top\phi(G(\boldsymbol{x}_i^S)) - \rho)\}, \rho\} = 1$[1]. The optimization problem (2) can be rewritten as follows:

$$\min_{w,\rho}\frac{1}{2}\|\mathbf{w}\|^2 \tag{3}$$

subject to

$$y_i(\mathbf{w}^\top\phi(G(\boldsymbol{x}_i^S)) - \rho) \geq 0,\ i = 1, ..., m$$
$$y_i(\mathbf{w}^\top\phi(G(\boldsymbol{x}_i^S)) - \rho) \geq 1,\ i = m+1, ..., N_S$$
$$\mathbf{w}^\top\phi(G(\boldsymbol{x}_i^T)) \geq \rho,\ i = 1, ..., N_T$$

We refer to the above model as a hard version of our proposed cross-domain kernel classifier. To derive the soft version (to let the model be applicable to the case where the data are linearly inseparable), inspired by [Schölkopf et al.(2001)], we extend the optimization problem in Eq. (3) by using the slack variables as follows:

$$\min_{\mathbf{w},\rho}(\frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{N_S+N_T}(\sum_{i=1}^{N_S}\xi_i^S + \lambda\sum_{i=1}^{N_T}\xi_i^T)) \tag{4}$$

subject to

[1]This assumption is feasible because if $(\mathbf{w}^*, \rho^*)$ is the optimal solution, $(k\mathbf{w}^*, k\rho^*)$ with $k > 0$ is also another optimal solution. Therefore, we can choose k to satisfy the assumption.

$$
\begin{aligned}
&y_i(\mathbf{w}^\top \phi(G(\boldsymbol{x}_i^S)) - \rho) \geq -\xi_i^S,\ i = 1, ..., m \\
&y_i(\mathbf{w}^\top \phi(G(\boldsymbol{x}_i^S)) - \rho) \geq 1 - \xi_i^S,\ i = m+1, ..., N_S \\
&\mathbf{w}^\top \phi(G(\boldsymbol{x}_i^T)) \geq \rho - \xi_i^T,\ i = 1, ..., N_T \\
&\xi_i^S \geq 0,\ i = 1, ..., N_S; \xi_i^T \geq 0,\ i = 1, ..., N_T.
\end{aligned}
$$

where $\lambda > 0$ is the trade-off hyper-parameter representing the weight of the information from the target domain contributing to the cross-domain kernel classifier.

We derive the primal form of the soft model optimization problem in Eq. (4) as follows:

$$
\min_{\mathbf{w},\rho} \mathcal{L}(G, \mathbf{w}, \rho) \tag{5}
$$

where we have defined

$$
\begin{aligned}
\mathcal{L}(G, \mathbf{w}, \rho) :=& \frac{1}{2} \|\mathbf{w}\|^2 + \frac{1}{N_S + N_T} \sum_{i=1}^{m} \max\{0, -z_i\} \\
&+ \frac{1}{N_S + N_T} \sum_{i=m+1}^{N_S} \max\{0, -z_i + 1\} \\
&+ \frac{\lambda}{N_S + N_T} \sum_{i=1}^{N_T} \max\left\{0, -\mathbf{w}^\top \phi(G(\boldsymbol{x}_i^T)) + \rho\right\}
\end{aligned}
$$

with $z_i = y_i \left(\mathbf{w}^\top \phi(G(\boldsymbol{x}_i^S) - \rho\right)$.

Random feature map. Deriving from the max-margin principle, in Eq. (5), we use the transformer ϕ to transform data from the latent space into a higher-dimensional space (i.e., the feature space), allowing for the discovery of complex patterns and relationships that might not be readily apparent in the original data space. This transformation is achieved using mathematical kernel functions (e.g., Gaussian kernel taking input vectors in the original space and returning the dot product of the vectors in the feature space [Hearst et al.(1998), Schölkopf and Smola(2002), Hofmann et al.(2008)]). However, the use of kernel functions can indeed result in a high-dimensional feature space, which can substantially increase the computational complexity and time required for training [Joachims(2006)].

To accelerate the training of the max-margin principle-based optimization problem as mentioned in Eq. (5), we use a random feature map [Rahimi and Recht(2008)](*i.e., helps accelerate the training of kernel machines by converting the training and evaluation of any kernel machine into the corresponding operations of a linear machine by mapping data into a relatively low-dimensional randomized feature space*) for the transformation ϕ to map the representations (e.g., $G(\boldsymbol{x}_i^S)$ and $G(\boldsymbol{x}_i^T)$) from the latent space to a random feature space. The formulation of ϕ on a specific $G(\boldsymbol{x}_i) \in \mathbb{R}^d$ is as follows:

$$
\phi(G(\boldsymbol{x}_i)) = [\frac{1}{\sqrt{K}} cos(\omega_k^\top G(\boldsymbol{x}_i), \frac{1}{\sqrt{K}} sin(\omega_k^\top G(\boldsymbol{x}_i)]_{k=1}^K
$$

where K consists of independent and identically distributed samples $\omega_1, ..., \omega_K \in \mathbb{R}^d$ which are the Fourier random elements. We note that the use of a random feature map ϕ [Rahimi and Recht(2008)] in conjunction with the cost-sensitive kernel machine of our proposed cross-domain kernel classifier as mentioned in Eq. (5) and a bidirectional recurrent neural network for the generator G allows us to conveniently do back-propagation when training our proposed approach.

Combining the optimization problems in Eqs. (1 and 5), we arrive at the final objective function:

$$
\mathcal{I}\ (G, D, \mathbf{w}, \rho) := \mathcal{L}(G, \mathbf{w}, \rho) + \alpha \mathcal{H}\ (G, D) \tag{6}
$$

where $\alpha > 0$ is the trade-off hyper-parameter. We seek the optimal generator G^*, domain discriminator D^*, the normal vector $\mathbf{w}^*$ and bias ρ^* by solving:

$$(G^*, \mathbf{w}^*, \rho^*) = \underset{G,\mathbf{w},\rho}{\operatorname{argmin}} \mathcal{I}\ (G, D, \mathbf{w}, \rho)$$

$$D^* = \underset{D}{\operatorname{argmax}}\ \mathcal{I}\ (G, D, \mathbf{w}, \rho)$$

The training algorithm of our proposed approach is shown in Algorithm 1.

Algorithm 1: The algorithm of our proposed method for cross-domain imbalanced software vulnerability detection.

Input: A labeled source domain dataset $S = \left\{\left(\boldsymbol{x}_1^S, y_1\right), \ldots, \left(\boldsymbol{x}_{N_S}^S, y_{N_S}\right)\right\}$ where $y_i \in \{-1, 1\}$ (i.e., 1: vulnerable code and -1: non-vulnerable code) and an unlabeled target domain dataset $T = \left\{\boldsymbol{x}_1^T, \ldots, \boldsymbol{x}_{N_T}^T\right\}$. The number of training iterations nt; the minibatch size m; the trade-off hyper-parameters α and λ.
We randomly partition the source domain S into the training set S_{train} (used to train the model) and the validation set S_{val} (used to save the best-trained model).
We randomly split the target domain T into the training set T_{train} (used to train the model) and the testing set T_{test} (used to evaluate the best-trained model).

1 Initialize the generator G, the cross-domain kernel classifier C, the discriminator D parameters with random weights θ_G, θ_C including ($\mathbf{w}$ and bias ρ), and θ_D respectively.

2 **for** $t = 1$ *to* nt **do**

3 Choose a minibatch of source domain samples $\left\{\left(\boldsymbol{x}_i^S, y_i^S\right)\right\}_{i=1}^m$ and target domain samples $\left\{\boldsymbol{x}_i^T\right\}_{i=1}^m$ randomly.

4 Update the generator G parameter (θ_G) and the cross-domain kernel classifier C parameter (θ_C) by minimizing the objective function $\mathcal{I}\ (G, D, \mathbf{w}, \rho)$ mentioned in Eq. (6) using the Adam optimizer.

5 Update the discriminator D parameter (θ_D) by maximizing the objective function $\mathcal{I}\ (G, D, \mathbf{w}, \rho)$ mentioned in Eq. (6) using the Adam optimizer.

6 **end**

Output: The optimal θ_G^*, θ_C^*, and θ_D^* parameters. After the training process, when the source and target domains are intermingled, we can transfer the trained cross-domain kernel classifier C to classify the data of the target domain. We evaluate the trained model's performance on the testing set T_{test} of the target domain T.

6 EXPERIMENTS

6.1 Experimental Design

The key goal of this experiment section is to evaluate the performance of our DAM2P and compare it with recent state-of-the-art cross-domain software vulnerability detection methods [Nguyen et al.(2019), Nguyen et al.(2020)] *and potential deep-domain adaptation approaches* [Ganin and Lempitsky(2015), Long et al.(2015), Nguyen et al.(2017), Shu et al.(2018), Chen et al.(2020), Le et al.(2021)] for cross-domain imbalanced software vulnerability detection.

We aim to address the following key Research Questions (RQs) in these experiments:

(RQ1) Can our DAM2P approach learn automatic features and exploit the imbalanced nature of source code data via deep domain adaptation and the max-margin principle to effectively perform transfer learning from imbalanced labeled source software projects to imbalanced unlabeled target software projects? Recent methods (e.g., [Nguyen et al.(2019), Nguyen et al.(2020)]) have been obtaining promising performances for cross-project software vulnerability detection (SVD); However, none of them exploit the imbalanced nature of source code projects for which the vulnerable data points are significantly minor compared to non-vulnerable ones. This may limit the capability of these methods in dealing with the cross-project SVD problem. In particular, without a robust capability to learn from small amounts of data (vulnerable data), SVD models may be disproportionately influenced by large amounts of data (non-vulnerable), thus potentially diminishing their ability to detect vulnerabilities in cross-domain vulnerability classification. The negative effect of this issue can be exhibited via the F1-measure of the models when applied to the target domain.

In this paper, in addition to exploiting deep domain adaptation with automatic representation learning for cross-project SVD, we propose a novel cross-domain kernel classifier leveraging the max-margin principle aiming to improve the capability of the transfer learning of software vulnerabilities from imbalanced labeled projects into imbalanced unlabeled ones. We investigate and prove that taking into account the imbalanced nature of source code data via the max-margin principle will help significantly improve the model performance in cross-project imbalanced SVD.

In real-world source code data, the vulnerable data and corresponding non-vulnerable data (e.g., functions or programs) can share common vulnerability-irrelevant source code statements while only a few core code statements cause associated data vulnerable instead of non-vulnerable. That poses a challenging problem for software vulnerability detection methods, especially in cross-domain vulnerability detection. This problem again highlights the necessity for vulnerability detection methods, especially in cross-domain vulnerability classification, to incorporate elegant mechanisms that effectively consider both vulnerable and non-vulnerable data when constructing decision boundaries.

(RQ2) Can our DAM2P approach successfully leverage the information from the imbalanced unlabeled target domain to further improve the model performance for cross-project imbalanced SVD? Our proposed approach can not only take into account the imbalanced nature of source code data but also leverage the information from the imbalanced unlabeled projects to improve the transfer learning capability from imbalanced labeled projects into imbalanced unlabeled ones. In this research question, we investigate the effectiveness of leveraging the information from the imbalanced unlabelled target domain in improving the capability of the transfer learning of software vulnerabilities from imbalanced labeled projects into imbalanced unlabeled projects.

(RQ3) Can techniques commonly used for solving the imbalanced dataset problem help to improve performance in the context of cross-domain imbalanced software vulnerability detection? Although achieving promising performances, none of the cross-domain SVD methods successfully exploit the imbalanced nature of source code projects, for which the vulnerable data points are significantly minor compared to non-vulnerable ones, to boost the performance on cross-domain software vulnerability detection. Without a robust capability to learn from minor data (vulnerable data), SVD models are likely to be overly influenced by major data (non-vulnerable), which can potentially diminish their ability to detect vulnerabilities. In this research question, we investigate if some commonly used techniques (e.g., Sampling and weighting [Chawla et al.(2002)], and Logit adjustment [Menon et al.(2021)]) addressing the imbalanced dataset problem can help to improve the baseline's performance to cross-project software vulnerability detection.

(RQ4) Can recent state-of-the-art methods in software vulnerability detection (SVD) be applied to solve the problem of cross-domain SVD?

Our study is for cross-domain software vulnerability detection. In addition to comparing our proposed method with the state-of-the-art cross-domain SVD approaches, we also want to investigate if recent effective work in conventional SVD, such as CodeBERT [Feng et al.(2020)] (a pre-trained model specializes in the programming language) and ReGVD [Nguyen et al.(2022b)] (an effective Graph neural network-based model for SVD), can produce results that accurately predict vulnerabilities across domains.

Experimental Datasets. We used the same several real-world source code datasets as studied in [Nguyen et al.(2019), Nguyen et al.(2020)]. These contain the source code of vulnerable functions (vul-funcs) and non-vulnerable functions (non-vul-funcs) obtained from five real-world software project datasets, namely FFmpeg (#vul-funcs: 187 and #non-vul-funcs: 5427), LibTIFF (#vul-funcs: 81 and #non-vul-funcs: 695), LibPNG (#vul-funcs: 43 and #non-vul-funcs: 551), VLC (#vul-funcs: 25 and #non-vul-funcs: 5548), and Pidgin (#vul-funcs: 42 and #non-vul-funcs: 8268).

Note that these real-world project (domain) datasets used in our experiments are extremely imbalanced. The number of vulnerable data in each project (domain) is only around 0.51% to 11.65% compared to the number of non-vulnerable data. Via our observation, in cross-domain vulnerability detection, within the same pair of the source and target domains, the fewer the number of vulnerable data compared to the number of non-vulnerable data, the more serious this problem may be. In reality, this problem can also happen across different pairs of the source and target domains.

In our experiments, to demonstrate the capability of our proposed method in transfer learning for cross-domain imbalanced software vulnerability detection (SVD) (i.e., transferring the learning of software vulnerabilities (SVs) from labeled projects to unlabelled projects belonging to different application domains), we used the multimedia application datasets (FFmpeg, VLC, and Pidgin) as the source domains, whilst the datasets (LibPNG and LibTIFF) from the image application domains were used as the target domains. It is worth noting that in the training process, we hide the labels of datasets from the target domains. We only use these labels in the testing phase to evaluate the models' performance.

Baselines. *The main baselines of our DAM2P method are some state-of-the-art end-to-end deep domain adaptation (DA) approaches for cross-domain SVD* including SCDAN [Nguyen et al.(2019)], Dual-GD-DDAN, Dual-GD-SDDAN [Nguyen et al.(2020)]. We also compare our method with other popular state-of-the-art domain adaptation approaches (i.e., most of the DA methods have been applied for vision data) including DDAN [Ganin and Lempitsky(2015)], MMD [Long et al.(2015)], D2GAN [Nguyen et al.(2017)], DIRT-T [Shu et al.(2018)], HoMM [Chen et al.(2020)], and LAMDA [Le et al.(2021)] as well as the state-of-the-art automatic feature learning for SVD, VulDeePecker [Li et al.(2018b)]. To the method operated via separated stages proposed by [Liu et al.(2020)], at present, we cannot compare to it due to the lack of the original data and completed reproducing source code from the authors.

VulDeePecker [Li et al.(2018b)] is an automatic feature learning method for SVD. The model employed a bidirectional recurrent neural network to take sequential inputs and then concatenated hidden units as inputs to a feedforward neural network classifier while the DDAN, MMD, D2GAN, DIRT-T HoMM, and LAMDA methods are the state-of-the-art deep domain adaptation models for computer vision proposed in [Ganin and Lempitsky(2015)], [Long et al.(2015)], [Nguyen et al.(2017)], [Shu et al.(2018)], [Chen et al.(2020)], and [Le et al.(2021)] respectively. Inspired by [Nguyen et al.(2019)], we borrowed the principles of these methods and refactored them using the CDAN architecture introduced in [Nguyen et al.(2019)] for cross-domain SVD. *It is worth noting that VulDeePecker is one of the state-of-the-art methods for SVD, but was not originally proposed*

for cross-domain SVD. In our paper, it is used to show that simply applying the classifier learned from labeled projects to classify the data from different unlabeled projects is not a good solution due to the shifted data distributions.

The SCDAN method [Nguyen et al.(2019)] can be considered as the first method that demonstrates the feasibility of deep domain adaptation for cross-domain SVD. Based on their proposed CDAN architecture, leveraging deep domain adaptation with automatic feature learning for SVD, the authors proposed the SCDAN method to exploit and utilize the information efficiently from unlabeled target domain data to improve the model performance. The Dual-GD-DDAN and Dual-GD-SDDAN methods were proposed in [Nguyen et al.(2020)] aiming to deal with the mode collapsing problem existing in SCDAN and other approaches using GAN as a principle to close the gap between the source and target domains in the latent space to further improve the transfer learning process for cross-domain SVD.

Data Processing and Embedding. We preprocess the source code datasets before inputting them into the deep neural networks (i.e., baselines and our proposed method). As shown in Figure 4, inspired by the baselines, we first standardize the source code by removing comments, blank lines, and non-ASCII characters. Secondly, we map user-defined variables to symbolic variable names (e.g., "*var1*", "*var2*") and user-defined functions to symbolic function names (e.g., "*func1*", "*func2*"). We also replace integer, real and hexadecimal numbers with a generic *number* token and strings with a generic *str* token. We then embed the source code statements into numeric vectors. For example, to the following code statement *"if(func2(func3(number,number),&var2) !=var10)*", we tokenize it to a sequence of code tokens (e.g., *if,(,func2,(,func3,(,number,number,),&,var2,),!=,var10,)*), construct the frequency vector of the statement information, and multiply this frequency vector by a learnable embedding matrix W^{si}.

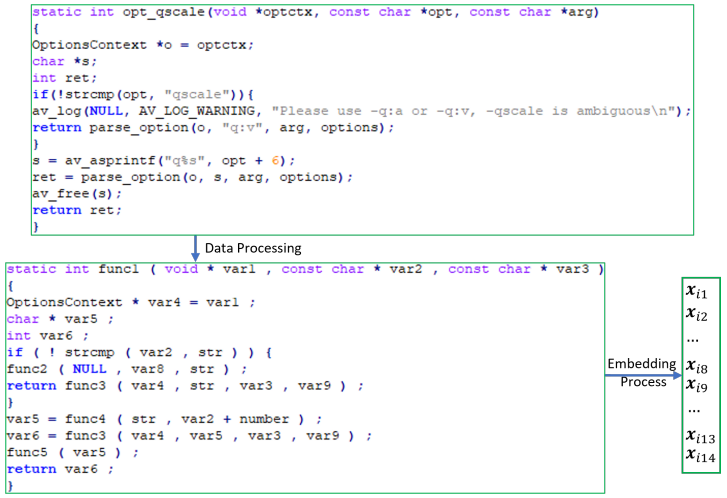


Fig. 4. An example of the overall procedure for data processing and embedding. We use a source code function in the C language programming from the FFmpeg project. After the data preprocessing step, we obtain a preprocessed function and then use the embedding process (a part of the model) to obtain the embedded vectors for the code statements of the function.

Note that as the baselines, to our proposed method, for handling the sequential properties of the data and to learn the automatic features of the source code functions, we also use a bidirectional recurrent neural network (bidirectional RNN) for both the source and target domains.

Model Configuration. For the baseline approaches including VulDeePecker [Li et al.(2018b)], and DDAN [Ganin and Lempitsky(2015)], MMD [Long et al.(2015)], D2GAN [Nguyen et al.(2017)], DIRT-T [Shu et al.(2018)], HoMM [Chen et al.(2020)], LAMDA [Le et al.(2021)] using the architecture CDAN proposed in SCDAN [Nguyen et al.(2019)], and Dual-GD-DDAN and Dual-GD-SDDAN [Nguyen et al.(2020)], and our proposed DAM2P method, we use one bidirectional recurrent neural network with LSTM [Hochreiter and Schmidhuber(1997)] cells where the size of hidden states is in $\{128, 256\}$ for the generator G while to the source classifier C used in the baselines and the domain discriminator D, we use deep feed-forward neural networks consisting of two hidden layers where the size of each hidden layer is equal to 300. We embed the statement information in the 150 dimensional embedding space.

To our proposed method, the trade-off hyper-parameters λ and α are in $\{10^{-3}, 10^{-2}, 10^{-1}\}$ and $\{10^{-2}, 10^{-1}, 10^{0}\}$, respectively, while the hidden size h is in $\{128, 256\}$. The dimension of random feature space $2K$ equals 1024. The length L of each function is padded or cut to 100 or less than 100 code statements (i.e., we base on the quantile values of the functions' length of each dataset to decide the length of each function). In particular, more than 96% of functions contain fewer than 100 code statements. Of the remaining 4% with over 100 statements, a manual examination reveals that nearly all of these functions exhibit critical code statements, those contributing to vulnerability, within their initial 100 code statements.

We employed the Adam optimizer [Kingma and Ba(2014)] with an initial learning rate of 10^{-3} while the mini-batch size is set to 100 for our proposed method and baselines. We split the data of the source domain into two random partitions containing 80% for training and 20% for validation. We also split the data of the target domain into two random partitions. The first partition contains 80% for training the models of MMD, D2GAN, DIRT-T, HoMM, LAMDA, DDAN, SCDAN, Dual-GD-DDAN, Dual-GD-SDDAN, and DAM2P without using any label information while the second partition contains 20% for testing the models. We additionally applied gradient clipping regularization to prevent the over-fitting problem in the training process of each model. We ran the corresponding model 5 times for each method and reported the averaged measures. We implemented all mentioned methods in Python using Tensorflow [Abadi et al.(2016)], an open-source software library for Machine Intelligence developed by the Google Brain Team, on an Intel E5-2680, having 12 CPU Cores at 2.5 GHz with 128GB RAM, integrated NVIDIA Tesla K80. Our released source code samples are publicly available at https://github.com/vannguyennd/dam2p.

6.2 Experimental Results

RQ1: Can our DAM2P approach learn automatic features and exploit the imbalanced nature of source code data via deep domain adaptation and the max-margin principle to effectively perform transfer learning from imbalanced labeled source software projects to imbalanced unlabeled target software projects?

Approach. We investigated the performance of our DAM2P method and compared it to the baselines. We note that the VulDeePecker method was only trained on the source domain data and then tested on the target domain data. The DDAN, MMD, D2GAN, DIRT-T, HOMM, LAMDA, SCDAN, Dual-GD-DDAN, Dual-GD-SDDAN, and DAM2P methods employed the target domain data without using any label information from this domain for deep domain adaptation.

The results in Table 1 show that our DAM2P method obtains a higher performance for most measures in the majority of cases of the source and target domains. DAM2P achieves the highest F1-measure for all pairs of the source and target domains. In general, our method obtains a higher performance on F1-measure from 1.83% to 6.25% compared to the second highest method in the

used source and target domains. For example, in the case of the source domain (FFmpeg) and the target domain (LibPNG), DAM2P obtains the F1-measure of *93.33%* compared with the F1-measure of 90.41%, 88.89%, 87.5%, 84.21%, 90.91%, 87.50%, 84.21%, 80%, 77.78% and 75% obtained by Dual-GD-SDDAN, Dual-GD-DDAN, SCDAN, DDAN, LAMDA, HOMM, DIRT-T, D2GAN, MMD and VulDeePecker, respectively.

The experimental results in Table 1 confirm the importance of addressing the imbalanced nature of source code data when developing a cross-domain vulnerability detection method. To improve the model's effectiveness in identifying both vulnerable and non-vulnerable data, it must robustly learn from both major (non-vulnerable) and minor (vulnerable) data. Furthermore, as indicated in Table 1, some baseline models demonstrate commendable recall results (e.g., over 80%) in certain scenarios within the source and target domains. However, these models tend to yield lower precision results in the corresponding cases, negatively affecting the corresponding F1-measure. This once again underscores the detrimental impact of the imbalanced nature of the source code data on the model performance where the baselines have not yet successfully addressed this issue.

It is important to highlight that for effective vulnerability detection, a method should exhibit high performances in both recall and precision, as indicated by the F1-measure, the harmonic mean of precision and recall. The F1-measure ensures a balanced assessment of a model's ability to correctly identify vulnerable instances while minimizing false positives. Ensuring high recall and high precision in security measures is crucial for thorough vulnerability detection and risk mitigation. While high recall tends to be prioritized (e.g., [Ami et al.(2024)]) to ensure the detection of the majority of vulnerabilities, thus reducing the risk of security breaches, maintaining a balance with high precision is also essential. This balance helps minimize false positives and optimize resource utilization, thereby preventing alert fatigue and avoiding disruption to legitimate data and activities. Therefore, a highly qualified vulnerability detection method needs to strike a high performance in both recall and precision. If a method exhibits high recall but low precision, it suggests a significant number of false positives where non-vulnerable data are incorrectly identified as vulnerable. This is likely to compromise the reliability of the vulnerability detection method.

The experimental results in Table 1 further reveal that the model performance (of our proposed DAM2P method and baselines) in cross-domain vulnerability detection is influenced not only by the ratio of vulnerable to non-vulnerable data in the source domain but also by the correlations (e.g., writing style) between source and target domain data. This explains why, in some pairs of source and target domains (e.g., Pidgin → LibTIFF and FFmpeg → LibTIFF) where the number of vulnerable data samples in a source domain is lower (e.g., Pidgin compared to FFmpeg), that may put more challenges on a model to distinguish between vulnerable and non-vulnerable data, the performance of the used models still vary, exhibiting lower, higher, or relatively consistent (e.g., on the pair Pidgin → LibTIFF compared to the pair FFmpeg → LibTIFF), across the used metrics including F1-measure, recall, and precision. Notably, our DAM2P method consistently achieves the highest F1-measure across all pairs of the source and target domains.

Visualization. We further demonstrate the efficiency of our proposed method in closing the gap between the source and target domains. We visualize the feature distributions of the source and target domains in the joint space using a 2D t-SNE [Maaten and Hinton(2008)] projection with perplexity equal to 30. In particular, we project the source domain and target domain data in the joint space (i.e., $G(\boldsymbol{x})$) into a 2D space without undertaking domain adaptation (using the VulDeePecker method) and with undertaking domain adaptation (using our DAM2P method).

In Figure 5, we present the results when performing domain adaptation from one software project (FFmpeg) to another (LibPNG). For the purpose of visualization, we select a random subset

Table 1. Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision, and F1-measure (F1) of VulDeePecker (VULD), MMD, D2GAN, DIRT-T, HOMM, LAMDA, DDAN, SCDAN, Dual-GD-DDAN (Dual-DDAN), Dual-GD-SDDAN (Dual-SDDAN) and DAM2P methods for predicting vulnerable and non-vulnerable functions on the testing set of the target domain (Best performance in **bold**).

Source → Target	Methods	FNR	FPR	Recall	Precision	F1
Pidgin→ LibPNG	VULD	42.86%	1.08%	57.14%	80%	66.67%
	MMD	37.50%	**0%**	62.50%	**100%**	76.92%
	D2GAN	33.33%	1.06%	66.67%	80%	72.73%
	DIRT-T	33.33%	1.06%	66.67%	80%	72.73%
	HOMM	14.29%	4.30%	85.71%	60.00%	70.59%
	LAMDA	**12.50%**	4.35%	87.50%	63.64%	73.68%
	DDAN	37.50%	**0%**	62.50%	**100%**	76.92%
	SCDAN	33.33%	**0%**	66.67%	100%	80%
	Dual-DDAN	33.33%	**0%**	66.67%	**100%**	80%
	Dual-SDDAN	22.22%	1.09%	77.78%	87.50%	82.35%
	DAM2P (ours)	**12.50%**	1.08%	**87.50%**	87.50%	**87.50%**
FFmpeg→ LibTIFF	VULD	43.75%	6.72%	56.25%	50%	52.94%
	MMD	28.57%	12.79%	71.43%	47.62%	57.14%
	D2GAN	30.77%	6.97%	69.23%	64.29%	66.67%
	DIRT-T	25%	9.09%	75%	52.94%	62.07%
	HOMM	37.50%	2.17%	62.50%	71.43%	66.67%
	LAMDA	37.50%	**1.09%**	62.50%	**88.33%**	71.42%
	DDAN	35.71%	6.98%	64.29%	60%	62.07%
	SCDAN	14.29%	5.38%	85.71%	57.14%	68.57%
	Dual-DDAN	**12.5%**	8.2%	**87.5%**	56%	68.29%
	Dual-SDDAN	35.29%	3.01%	64.71%	73.33%	68.75%
	DAM2P (ours)	14.29%	8.14%	85.71%	63.16%	**72.73%**
FFmpeg→ LibPNG	VULD	25%	2.17%	75%	75%	75%
	MMD	12.5%	3.26%	87.5%	70%	77.78%
	D2GAN	14.29%	2.17%	85.71%	75%	80%
	DIRT-T	15.11%	2.2%	84.89%	80%	84.21%
	HOMM	**0%**	2.15%	**100%**	77.78%	87.50%
	LAMDA	16.67%	**0.%**	83.33%	**100%**	90.91%
	DDAN	**0%**	3.26%	**100%**	72.73%	84.21%
	SCDAN	12.5%	1.08%	87.5%	87.5%	87.5%
	Dual-DDAN	**0%**	2.17%	**100%**	80%	88.89%
	Dual-SDDAN	17.5%	**0%**	82.5%	**100%**	90.41%
	DAM2P (ours)	**0%**	1.07%	**100%**	87.50%	**93.33%**
VLC→ LibPNG	VULD	57.14%	1.08%	42.86%	75%	54.55%
	MMD	45%	4.35%	55%	60%	66.67%
	D2GAN	28.57%	4.3%	71.43%	55.56%	62.5%
	DIRT-T	50%	1.09%	50%	80%	61.54%
	HOMM	42.86%	**0%**	57.14%	**100%**	72.73%
	LAMDA	28.57%	1.08%	71.43%	83.33%	76.92%
	DDAN	33.33%	2.20%	66.67%	75%	70.59%
	SCDAN	33.33%	1.06%	66.67%	80%	72.73%
	Dual-DDAN	28.57%	2.15%	71.43%	71.43%	71.43%
	Dual-SDDAN	**11.11%**	4.39%	**88.89%**	66.67%	76.19%
	DAM2P (ours)	33.33%	**0%**	66.67%	**100%**	**80%**
Pidgin→ LibTIFF	VULD	35.29%	8.27%	64.71%	50%	56.41%
	MMD	30.18%	12.35%	69.82%	50%	58.27%
	D2GAN	40%	7.95%	60%	60%	60%
	DIRT-T	38.46%	8.05%	61.54%	53.33%	57.14%
	HOMM	20%	9.41%	80%	60%	68.57%
	LAMDA	30%	4.44%	70%	63.64%	66.67%
	DDAN	27.27%	8.99%	72.73%	50%	59.26%
	SCDAN	30%	5.56%	70%	58.33%	63.64%
	Dual-DDAN	29.41%	6.76%	70.59%	57.14%	63.16%
	Dual-SDDAN	37.5%	**2.98%**	62.5%	**71.43%**	66.67%
	DAM2P (ours)	**7.69%**	9.20%	**92.31%**	60%	**72.73%**

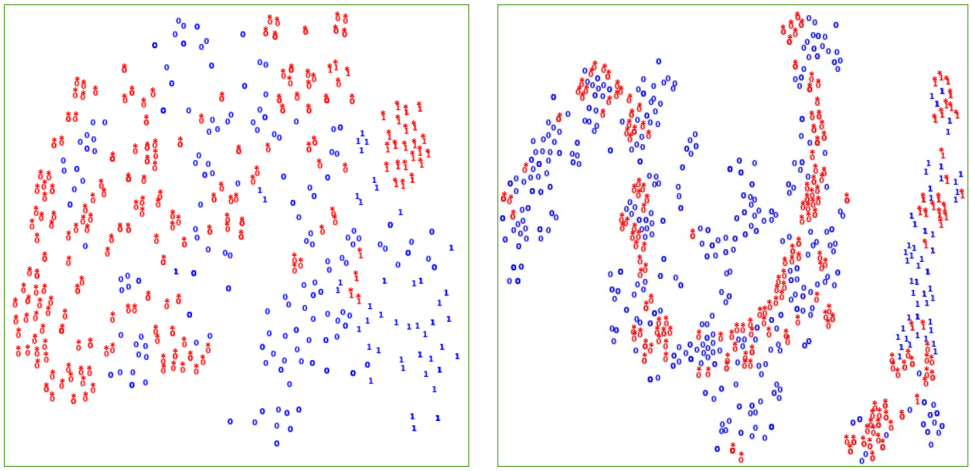


Fig. 5. A 2D t-SNE projection for the case of the FFmpeg → LibPNG without undertaking domain adaptation (the left-hand figure, using VulDeePecker) and undertaking domain adaptation (the right-hand figure, using our proposed DAM2P method). The blue points and the red points (with a special token ∗ on top of each point) represent the source and target domains in the joint space respectively. Data points labeled 0 stand for non-vulnerable samples and data points labeled 1 stand for vulnerable samples. *It is noted that our method can not only successfully bridge the gap between the source and target domains but also be able to distinguish the non-vulnerable and vulnerable data effectively*.

of the source project against the entire target project. As shown in Figure 5, without undertaking domain adaptation (VulDeePecker) the blue points (the source domain data) and the red points (the target domain data) are almost separate while with undertaking domain adaptation the blue and red points intermingled as expected. Furthermore, **we observe that the mixing-up level of the source domain and target domain data using our DAM2P method is significantly high. In particular, the source and target domains are mixed while the vulnerable and non-vulnerable data from both domains are separated**.

With the use of the max-margin principle to find the hyperplane that maximizes the margin between classes in the feature space, the decision boundary is placed as far away from the nearest data points of each class as possible. In real-world scenarios, especially in complex datasets, it is common to have a mixture of samples from different classes on or near the margin. This occurs because the margin is determined by the most challenging instances to classify, which are often the ones closest to the decision boundary. These instances can represent ambiguous or overlapping regions between classes, where the model is uncertain about the correct classification. Therefore, it can be usual to observe a mixture of benign and vulnerable samples near the margin. Our qualitative results, illustrated in Figure 5, also reveal a mixture of benign and vulnerable footpaths, with some vulnerable and non-vulnerable data points appearing close to each other. The visualizations show that our method successfully bridges the gap between the source and target domains while effectively distinguishing between non-vulnerable and vulnerable data.

Answers to RQ1: The quantitative experimental results in Table 1 on five main measures (i.e., false negative rate (FNR), false positive rate (FPR), Recall, Precision, and F1-measure (F1)) and the qualitative results in Figure 5 show the superiority of our DAM2P method in achieving high performances for cross-project imbalanced SVD on the used real-world datasets over the baselines.

RQ2: Can our DAM2P approach successfully leverage the information from the imbalanced unlabeled target domain to further improve the model performance for cross-project imbalanced SVD?

Approach. We aim to further demonstrate the efficiency of our DAM2P method in transferring the learning of vulnerabilities from imbalanced labeled source domains to other imbalanced unlabeled target domains as well as the superiority of our novel cross-domain kernel classifier in our DAM2P method for learning and separating vulnerable and non-vulnerable data. In particular, in this study, we not only prove the effectiveness of leveraging the information from the imbalanced unlabelled target domain in improving the capability of the transfer learning but also demonstrate that bridging the discrepancy gap in the latent space and using the max-margin cross-domain kernel classifier are complementary to boost the DA performance with imbalanced nature.

Table 2. Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision, and F1-measure (F1) of five cases including (i, VulDeePecker), (ii, DDAN), (iii, Kernel-Source denoted by Kernel-S), (iv, Kernel-Source-Target denoted by Kernel-ST), and (v, DAM2P) for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain (Best performance in **bold**).

Source → Target	Methods	FNR	FPR	Recall	Precision	F1
FFmpeg → LibTIFF	VulDeePecker	43.75%	6.72%	56.25%	50%	52.94%
	DDAN	35.71%	6.98%	64.29%	60%	62.07%
	Kernel-S	30%	**5.56%**	70%	58.33%	63.63%
	Kernel-ST	25%	5.68%	75%	**64.29%**	69.23%
	DAM2P (ours)	**14.29%**	8.14%	**85.71%**	63.16%	**72.73%**
FFmpeg → LibPNG	VulDeePecker	25%	2.17%	75%	75%	75%
	DDAN	**0%**	3.26%	**100%**	72.73%	84.21%
	Kernel-S	**0%**	4.39%	**100%**	69.23%	81.81%
	Kernel-ST	**0%**	3.26%	**100%**	72.72%	84.21%
	DAM2P (ours)	**0%**	**1.07%**	**100%**	**87.50%**	**93.33%**

We conduct experiments on two pairs of the source and target domains including FFmpeg → LibTIFF and FFmpeg → LibPNG. We consider five cases in which we start from the *blank case (i, VulDeePecker)* without bridging the gap and cross-domain kernel classifier. We then only add the GAN term to bridge the discrepancy gap in the *second case (ii, DDAN)*. In the *third case (iii, Kernel-Source)*, we only apply the max-margin principle for the source domain, while applying the max-margin principle for the source and target domains in the *fourth case (iv, Kernel-Source-Target)*. Finally, in the *last case (v, DAM2P)*, we simultaneously apply the bridging term and the max-margin terms for the source and target domains.

The results in Table 2 shows that the max-margin and bridging terms help to boost the domain adaptation performance. Moreover, applying the max-margin term to both the source and target domains improves the performance compared to applying it to only the source domain. Last but not least, bridging the discrepancy gap term in cooperation with the max-margin term significantly improves the domain adaptation performance.

Answers to RQ2: The experimental results (in Table 2) show a considerable increase in the model performance when simultaneously applying the bridging term and the max-margin terms for the source and target domains compared to cases without combining these terms. The results also show an improvement when harnessing the information from the imbalanced unlabelled target domain for learning the cross-domain classifier to enhance the capability of transfer learning.

RQ3: Can techniques commonly used for solving the imbalanced dataset problem help to improve performance in the context of cross-domain imbalanced software vulnerability detection?

Sampling and weighting, and Logit adjustment. Sampling and weighting are well-known as simple and heuristic methods to deal with imbalanced datasets. However, as mentioned by [Lin et al.(2017), Cui et al.(2019)], these methods may have some limitations, for example, i) Sampling may either introduce large amounts of duplicated samples, which slows down the training and makes the model susceptible to overfitting when oversampling, or discarding valuable examples that are important for feature learning when undersampling, and ii) To the highly imbalanced datasets, directly training the model or weighting (e.g., inverse class frequency or the inverse square root of class frequency) cannot yield satisfactory performance.

[Menon et al.(2021)] recently proposed a novel statistical framework for solving the imbalanced (long-tailed) label distribution problem. Specifically, the framework, revisiting the idea of *logit adjustment* based on the label frequencies, encourages a large relative margin between logits of the rare positive labels versus the dominant negative labels.

To investigate the efficiency of these methods (i.e., *sampling and weighting, and logit adjustment*) when applying to the baselines in the context of cross-domain imbalanced software vulnerability detection (SVD), we conduct an experiment on two pairs of the source and target domains (i.e., FFmpeg → LibTIFF and FFmpeg → LibPNG) for four main baselines including the DDAN, SCDAN, Dual-GD-DDAN, and Dual-GD-SDDAN methods using (i) the oversampling technique based on SMOTE [Chawla et al.(2002)] (i.e., used to create balanced datasets), and (ii) *logit adjustment* (LA) used in [Menon et al.(2021)].

Table 3. Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision, and F1-measure (F1) of DDAN, SCDAN, Dual-GD-DDAN (Dual-DDAN), and Dual-GD-SDDAN (Dual-SDDAN) methods in three cases of with using oversampling (w/ OS), using LA (w/ LA) and without using (oversampling or LA) (w/o (OS or LA)) for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain. We denote Source → Target by S → T.

S → T	Methods	FNR	FPR	Recall	Precision	F1
FFmpeg → LibTIFF	DDAN w/ OS	21%	12.79%	78.57%	50%	61.11%
	DDAN w/ LA	14.29%	15.11%	85.71%	48%	61.53%
	DDAN w/o (OS or LA)	35.71%	6.98%	64.29%	60%	**62.07%**
	SCDAN w/ OS	25%	5.43%	75%	54.55%	63.16%
	SCDAN w/ LA	11.11%	8.8%	88.89%	50%	64%
	SCDAN w/o (OS or LA)	14.29%	5.38%	85.71%	57.14%	**68.57%**
	Dual-DDAN w/ OS	25%	6.72%	75%	57.14%	64.87%
	Dual-DDAN w/ LA	35.29%	4.50%	64.71%	64.71%	64.71%
	Dual-DDAN w/o (OS or LA)	12.5%	8.2%	87.5%	56%	**68.29%**
	Dual-SDDAN w/ OS	16.67%	9.1%	83.33%	56%	67%
	Dual-SDDAN w/ LA	43.75%	1.70%	56.25%	90%	**69%**
	Dual-SDDAN w/o (OS or LA)	35.29%	3.01%	64.71%	73.33%	68.75%
FFmpeg → LibPNG	DDAN w/ OS	28.57%	0%	71.43%	100%	83.33%
	DDAN w/ LA	25%	0%	75%	100%	**85.71%**
	DDAN w/o (OS or LA)	0%	3.26%	100%	72.73%	84.21%
	SCDAN w/ OS	25%	0%	75%	100%	85.71%
	SCDAN w/ LA	0%	2.17%	100%	80%	**88.89%**
	SCDAN w/o (OS or LA)	12.5%	1.08%	87.5%	87.5%	87.5%
	Dual-DDAN w/ OS	14.29%	1.08%	85.71%	85.71%	85.71%
	Dual-DDAN w/ LA	0%	2.15%	100%	77.78%	87.5%
	Dual-DDAN w/o (OS or LA)	0%	2.17%	100%	80%	**88.89%**
	Dual-SDDAN w/ OS	12.5%	2.17%	87.5%	77.78%	82.35%
	Dual-SDDAN w/ LA	11.11%	1.1%	88.89%	88.89%	88.89%
	Dual-SDDAN w/o (OS or LA)	17.5%	0%	82.5%	100%	**90.41%**

The results in Table 3 show that using the oversampling technique cannot help to improve these baseline models' performance. In particular, the performance of these baselines without using oversampling is always higher than using oversampling on the used datasets in F1-measure (F1), the most important measure used in SVD. This experiment supports our conjecture that in the context of imbalanced domain adaptation when moving target representations to source representations in the latent space to bridge the gap, oversampling the minority class (i.e., vulnerable class) might increase the chance of wrongly mixing up vulnerable representations of the target domain and non-vulnerable representations of the source domain, hence leading to a reduction in performance. In our approach, with the support of the max-margin principle, we keep the vulnerable and non-vulnerable representations distant as much as possible when bridging the gap between these representations in the latent space.

Furthermore, the results in Table 3 indicate that using LA can help slightly improve the model's performance on some cases of the baselines. In particular, LA increases the performance of Dual-SDDAN on FFmpeg → LibTIFF as well as DDAN and SCDAN on FFmpeg → LibPNG compared to these methods without using LA. However, to DDAN, SCDAN and Dual-DDAN on FFmpeg → LibTIFF as well as Dual-DDAN and Dual-SDDAN on FFmpeg → LibPNG, LA cannot help improve these models' performance. In conclusion, using LA can help increase the baseline's performance in some cases of the used datasets in F1-measure compared to these cases without using LA or using the oversampling technique. However, similar to the oversampling technique, in most cases mentioned in Table 3, LA cannot help improve the baselines' performance. Furthermore, in the cases where LA helps increase the baseline models' performance, our proposed method's performance (mentioned in Table 1) is still significantly higher.

Answers to RQ3: In general, the experimental results (in Table 3) show that commonly used techniques (e.g., oversampling (SMOTE) and LA) for solving the imbalanced dataset problem cannot help to improve the baselines' performance in the context of imbalanced cross-domain software vulnerability detection in most cases of the source and target domains. In the cases where LA helps increase the baseline's performance, our method's performance (mentioned in Table 1) is still significantly higher.

RQ4: Can recent state-of-the-art methods in software vulnerability detection (SVD) be applied to solve the problem of cross-domain SVD?

Approach. We investigated the capability of recent state-of-the-art methods in software vulnerability detection (SVD) including **CodeBERT** [Feng et al.(2020)] (a pre-trained model specializes in the programming language) and **ReGVD** [Nguyen et al.(2022b)] (an effective Graph neural network-based model for SVD) on the problem of cross-domain SVD. Like the VuldeePecker [Li et al.(2018b)] approach, these methods are well-known as effective and state-of-the-art solutions for SVD but were not originally designed for cross-domain SVD. In this experiment, we further demonstrate the limitation of simply applying the classifier learned from labeled projects to classify the data from different unlabeled projects due to the shifted data distributions. It indicates the need for cross-domain software vulnerability detection approaches. We used the recommendation settings and implemented the CodeBERT and ReGVD methods following the source code samples published by the authors. Note that the ReGVD and CodeBERT methods were trained and fine-tuned, respectively, on the source domains and then tested on the target domains.

We conduct the experiment on two pairs of the source and target domains (i.e., FFmpeg → LibTIFF and FFmpeg → LibPNG). The experimental results in Table 4 show that these methods cannot work well for the cross-domain imbalanced SVD. They achieve much lower performances for all the used

measures, especially CodeBERT, compared to the state-of-the-art methods in cross-domain SVD and our proposed method (mentioned in Table 1).

Table 4. Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision, and F1-measure (F1) of the CodeBERT and ReGVD methods for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain.

Source → Target	Methods	FNR	FPR	Recall	Precision	F1
FFmpeg → LibTIFF	CodeBERT	86.56%	12.01%	13.44%	23.21%	17.03%
	ReGVD	77.83%	10.08%	22.17%	37.26%	27.80%
FFmpeg → LibPNG	CodeBERT	98.59%	1.76%	1.40%	15.04%	2.57%
	ReGVD	33.61%	17.37%	66.39%	45.83%	54.23%

Answers to RQ4: The results in Table 4 again show the limitation of the state-of-the-art SVD when being applied to solve the problem of cross-domain SVD. Simply applying the classifier learned from labeled projects to classify the data from different unlabeled projects is not a good solution because of the shifted data distributions. That shows the need for cross-domain software vulnerability detection approaches.

Additional ablation studies for RQ4 and RQ1.

Would CodeBERT and ReGVD exhibit improved performances if tuned on domain-invariant features extracted through domain adaptation? To assess this, we conducted an additional experiment on two pairs of source and target domains (FFmpeg to LibPNG and FFmpeg to LibTIFF) in two consecutive steps. The first step focused solely on domain-invariant feature learning, while the second step involved fine-tuning CodeBERT and training ReGVD using the features obtained from the first step.

Table 5. Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision, and F1-measure (F1) of the CodeBERT and ReGVD methods for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain when they are tuned on the domain invariant features extracted from domain adaptation.

Source → Target	Methods	FNR	FPR	Recall	Precision	F1
FFmpeg → LibTIFF	CodeBERT	40%	8.63%	60%	20%	30%
	ReGVD	74.67%	11.87%	25.33%	36.55%	29.92%
FFmpeg → LibPNG	CodeBERT	0.0%	13.51%	100%	6.25%	11.76%
	ReGVD	30.86%	15.47%	69.14%	49.77%	57.88%

The results in Table 5 indicate that this approach helps to improve the performance of CodeBERT and ReGVD; however, it is still ineffective in assisting both CodeBERT and ReGVD in addressing the cross-domain imbalanced SVD, as these methods still exhibit low performances in terms of the F1-measure, especially when compared to the cross-domain SVD methods where the process of learning invariant features and the classifier are operated in the designed united framework to support each other to obtain the best performance for cross-domain SVD.

Would the initial representations be derived from pre-trained models (e.g., CodeBERT and UnixCoder), with the addition of a Max-Margin Classifier on top to assist the DAM2P method in addressing cross-domain imbalanced SVD?. To assess this experiment, we replace the bidirectional recurrent neural network (bidirectional RNN) used in the generator G with the pre-trained CodeBert and UnixCoder before applying the GAN principle in the latent space for bridging the gap between the source and target domains. We conduct a corresponding experiment on two pairs of

the source and target domains (including FFmpeg to LibPNG and FFmepg to LibTIFF) to investigate if this approach is an effective solution for cross-domain SVD. The results in Table 6 imply that this approach is not an effective way to assist the DAM2P method in addressing cross-domain imbalanced SVD, as the corresponding model performs poorly in terms of F1-measure.

Table 6. Performance results in terms of false negative rate (FNR), false positive rate (FPR), Recall, Precision, and F1-measure (F1) of the the DAM2P methods for predicting vulnerable and non-vulnerable code functions on the testing set of the target domain when the initial representations and the representation learning part are taken from pre-trained models such as CodeBERT and UnixCoder.

Source → Target	Pre-trained models	FNR	FPR	Recall	Precision	F1
FFmpeg → LibTIFF	CodeBERT	0.0%	100%	100%	11.33%	20.36%
	UnixCoder	29.41%	41.30%	70.59%	17.39%	27.91%
FFmpeg → LibPNG	CodeBERT	0.0%	100%	100%	10%	18.18%
	UnixCoder	30.77%	34.31%	69.23%	20.45%	31.58%

Via our observations, there may be some potential reasons why the initial representations and the representation learning part taken from pre-trained large language models (LLMs) do not work well on downstream tasks (e.g., cross-domain imbalanced SVD) such as (i) Domain shift: significant differences between the domain of the pre-training data and the domain of the downstream task can hinder transferability. The model may struggle to adapt if the task involves a different context or domain, (ii) Task complexity: some downstream tasks may be inherently more complex or have different patterns compared to the pre-training objectives. The model may not be able to adapt well to tasks with distinct complexities, and (iii) Lack of task-specific knowledge: LLMs are trained on a vast range of data, but they may not have specific knowledge about the nuances of particular tasks. Fine-tuning helps, but it might not be enough for highly specialized tasks.

6.3 Hyper-parameter Sensitivity

We discuss the correlation between important hyper-parameters (including the λ, α, and h (the size of hidden states in the bidirectional neural network)) and the F1-measure of our proposed DAM2P method. Our experiments found that the trade-off hyper-parameters λ and α are in $\{10^{-3}, 10^{-2}, 10^{-1}\}$ and $\{10^{-2}, 10^{-1}, 10^{0}\}$, respectively, while the hidden size h is in $\{128, 256\}$. It is worth noting that we use the commonly used values for the trade-off hyper-parameters (λ and α) representing for the weights of different terms mentioned in Eq. (6) and the hidden size h. In order to study the impact of the hyper-parameters on the performance of the DAM2P method, we use a wider range of values for λ, α, and h. In this ablation study, the trade-off parameters λ and α are in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$ while the hidden size h is in $\{32, 64, 128, 256, 512, 1024\}$.

We investigate the impact of λ, α, and h hyper-parameters on the performance of the DAM2P method on five pairs of the source and target domains including FFmpeg to LibPNG, FFmpeg to LibTIFF, Pidgin to LibPNG, Pidgin to LibTIFF, and VLC to LibPNG. As shown in Figures (6, 7, and 8), we observe that the appropriate values to the hyper-parameters used in the DAM2P model, in order to obtain the best model's performance, should be in from 10^{-4} to 10^{-2}, from 10^{-3} to 10^{-1}, and from 64 to 256 for λ, α, and h respectively. In particular, for the hidden size h, if we use too small values (e.g., ≤ 32) or too high values (e.g., ≥ 1024), the model might encounter underfitting or overfitting problems respectively. The model's performance on λ (i.e., representing the weight of the information from the target domain contributing to the cross-domain kernel classifier during the training process) shows that we should not set the value of λ equal or higher than 1.0 (i.e., used for the weight of the information from the source domain), and the value of λ should be higher than 10^{-4} to make sure that we use enough information of the target domain in the training process to improve the cross-domain kernel classifier.

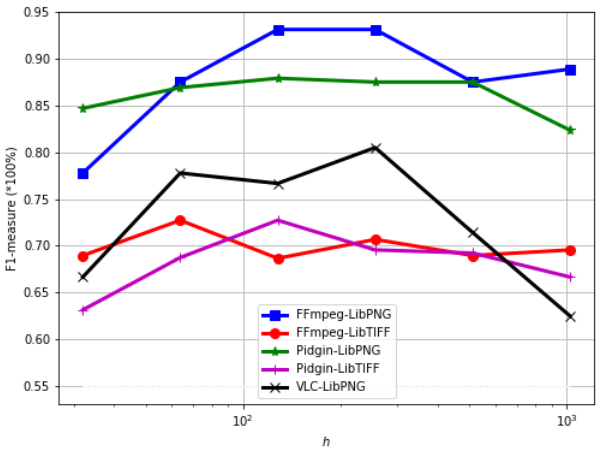


Fig. 6. The correlation between h and F1-measure of our proposed DAM2P method.

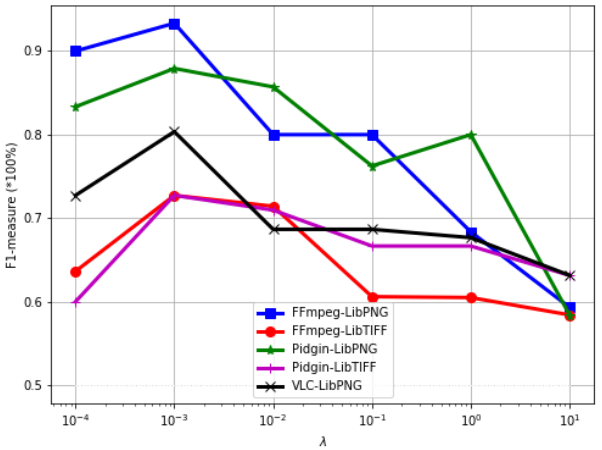


Fig. 7. The correlation between λ and F1-measure of our proposed DAM2P method.

6.4 Threats to Validity

Construct Validity: Key construct validity threats are if our assessments of the methods demonstrate the ability for cross-project software vulnerability detection (SVD). The main purpose of our DAM2P method is for cross-project SVD solving two crucial issues in SVD including i) learning automatic representations to improve the predictive performance of SVD, and ii) coping with the scarcity of labeled vulnerabilities in projects that require the laborious labeling of code by experts. To evaluate the performance of our DAM2P method and baselines, we use five main measures, including false negative rate (FNR), false positive rate (FPR), Recall, Precision, and F1-measure (F1), widely used in SVD [Li et al.(2016), Li et al.(2018a), Nguyen et al.(2019)].

Internal Validity: Key internal validity threats are relevant to the choice of hyper-parameter settings (i.e., optimizer, learning rate, number of layers in deep neural networks, etc.). It is worth noting that finding a set of optimal hyperparameter settings of deep neural networks is expensive due to a large number of trainable parameters. To train our method, we only use the common and default values of hyper-parameters such as using Adam optimizer and the learning rate equals 10^{-3}.

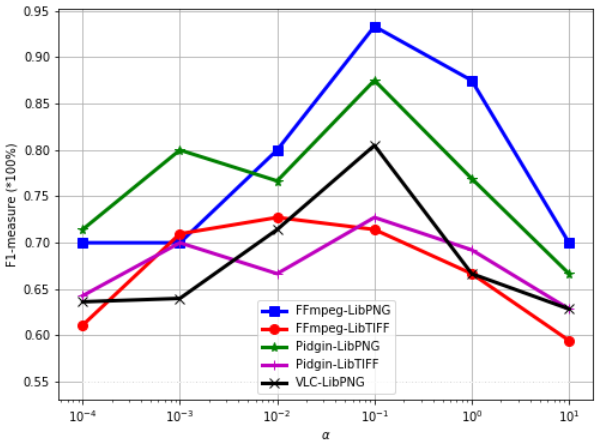


Fig. 8. The correlation between α and F1-measure of our proposed DAM2P method.

We also report the hyperparameter settings in the released reproducible source code to support future replication studies.

In our study, as the baselines, we use a bi-directional long short-term memory (LSTM) for the feature extraction network (i.e., the generator G). A bi-directional LSTM is designed to capture long-term dependencies in sequential data, which often requires processing a large number of time steps. This can lead to increased training time, especially when dealing with long sequences in huge datasets. In our proposed method, the embedding, feature extraction (generator G), domain discriminator D, and the cross-domain kernel classifier C are operated in a united framework. These parts link and support each other in obtaining the optimal solution to solving the problem. Our cross-domain kernel classifier based on the max-margin principle (which is well-known for achieving high generalization performance even with a relatively small number of data points) can help our model faster to coverage to the optimal decision boundary reducing the training time.

External Validity: Key external validity threats include whether our proposed method will generalize to other vulnerabilities (i.e., vulnerabilities can be of various types and natures presenting in both source domains and target domains although written in different ways, e.g., using different structures or variable names) and whether they will work on other source code datasets. We mitigated this problem by using five real-world source code datasets, namely FFmpeg, LibTIFF, LibPNG, VLC, and Pidgin.

6.5 Future study

As mentioned, in the scope of our paper, different software domains are those that come from different software applications (e.g., multimedia applications and image applications) written in the same C/C++ programming language. These domains are written in different ways (e.g., using different structures or variable names); however, they share the same nature (types) of code vulnerabilities. This property lets our proposed method and the baselines be deployed realistically for cross-domain software vulnerability detection.

Based on our observations, we have noted that source code data from various domains may be composed in different programming languages, such as C/C++ and Java. However, what is crucial is that they share common characteristics in terms of code vulnerabilities, such as similar semantic relationships. We plan to delve deeper into this aspect in our future studies.

In addition, to both conventional software vulnerability detection (training and testing phrases for data from the same project) and cross-domain vulnerability detection methods (the practical problem studied in our paper), the vulnerabilities from the same project testing sets (for conventional SVD) or from the target domain testing sets (for cross-domain SVD) are assumed to be similar or the same from the training sets (of the source domain), respectively.

The research for dealing with new vulnerabilities emerging in the target domains could be a focus of our future research. To the best of our knowledge, in domain adaptation, there are no effective methods that can deal with the case of new labels (e.g., vulnerabilities) appearing from the target domains in the context of unsupervised domain adaptation (during the training process, we do not use any label information from the target domains for deep domain adaptation). In our view, tackling new labels in the target domains, as opposed to the labels from the source domains, requires research that utilizes both data and labels from the target domains.

7 ADDITIONAL RELATED BACKGROUND

7.1 One-class support vector machine

Schölkopf and colleagues [Schölkopf et al.(2001)] proposed an approach to customize Support Vector Machines (SVM) for one-class classification. Their method involves first applying a kernel transformation to the features. Next, they designate the origin as the sole representative of the second class. By introducing "relaxation parameters" they create a margin to separate the one class's representation from the origin. Subsequently, they apply standard two-class SVM techniques to the problem.

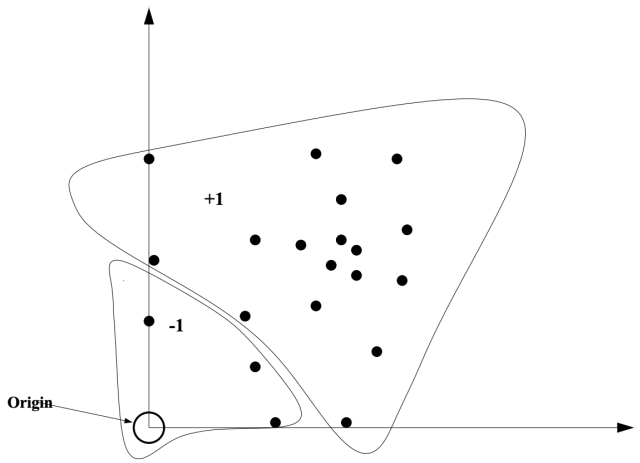


Fig. 9. One-Class SVM Classifier. The origin is the only original member of the second class.

Assuming that there is a dataset sampled from an underlying probability distribution represented by P, one needs to estimate a "simple" subset S of the input space such that the probability that a test point from P lies outside of S is bounded by a prior specified $v \in (0, 1)$. The solution for this problem is achieved by estimating a function f which is positive on S and negative on the complement $\bar{S}$. In other words, Schölkopf and colleagues introduced an algorithm that returns a function f taking the value +1 in a "small" region capturing most of the data vectors, and -1 elsewhere. The algorithm can be summarized as mapping the data into a feature space H using an

appropriate kernel function, and then trying to separate the mapped vectors from the origin with a maximum margin where $f(\boldsymbol{x}) = +1$ if $\boldsymbol{x} \in S$ and $f(\boldsymbol{x}) = -1$ if $\boldsymbol{x} \in \bar{S}$.

Let $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_l$ be training examples belonging to one class X, a compact subset of R^N. Let $\phi : X \rightarrow H$ be a kernel map that transforms the training examples to another space (e.g., the feature space). Then, to separate the data set from the origin, one needs to solve the following quadratic programming problem:

$$\min_{\mathbf{w},\rho}(\frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{vl}\sum_{i=1}^{l}\xi_i - \rho) \tag{7}$$

subject to

$$\mathbf{w}^\top\phi(\boldsymbol{x}_i) \geq \rho - \xi_i,\ i = 1, ..., l \quad \xi_i \geq 0$$

If $\mathbf{w}$ and ρ solve this problem, then the decision function $f(\boldsymbol{x}) = sign(\mathbf{w}^\top\phi(\boldsymbol{x}) - \rho)$ will be positive for most examples $\boldsymbol{x}_i$ contained in the training set. A visualization of the one-class support vector machine is depicted in Figure 9.

7.2 Recurrent neural networks

Recurrent neural networks (RNNs) [Rumelhart et al.(1986)], a class of deep neural networks (DNNs), are specialized for sequential data (e.g., time series, sentences, documents, or audio samples). RNNs are extremely useful for natural language processing (NLP) systems [Sutskever et al.(2014)] such as automatic translation, speech-to-text, and sentiment analysis. Leveraging the idea of sharing parameters across different parts of a model, an RNN can extend and apply to data of different forms and generalize across them. An RNN is similar to a DNN, except it has backward connections. A visualization of an RNN's architecture is depicted in Figure 10 (the left-hand figure). At each time step t, the state of a recurrent neuron (i.e., a hidden state $\boldsymbol{h}_t$) will receive the input vector $\boldsymbol{x}_t$ as well as the state vector from the previous step $t-1$ (i.e., $\boldsymbol{h}_{t-1}$) to obtain the state vector $\boldsymbol{h}_t$.

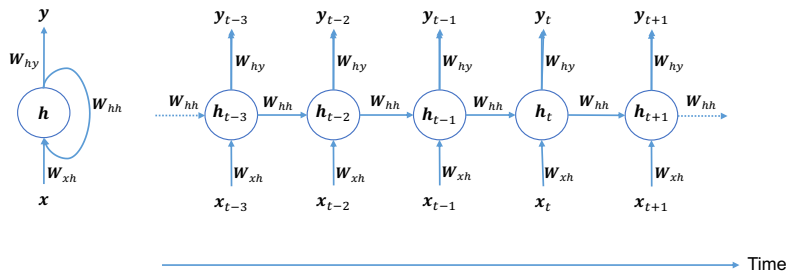


Fig. 10. An architecture of a recurrent neural network (RNN) with the outputs $\boldsymbol{y}$ and the hidden states $\boldsymbol{h}$ of recurrent neurons.

In particular, we have:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

We can unroll the RNN network through time to gain a new visualization as depicted in Figure 10 (the right-hand figure). Each recurrent neuron has two relevant input weights. One is for the input vector $\boldsymbol{x}_t$, and the other is for the state vector $\boldsymbol{h}_{t-1}$ of the previous time step $t-1$. At the time step t, if we denote the weight from the input vector $\boldsymbol{x}_t$ to the state $\boldsymbol{h}_t$ of the current recurrent neuron by $\mathbf{W}_{xh}$ and the weight from the state $\boldsymbol{h}_{t-1}$ of the previous recurrent neuron to the state $\boldsymbol{h}_t$

of the current recurrent neuron by $\mathbf{W}_{hh}$, the state $\boldsymbol{h}_t$ of the current recurrent neuron is computed as follows:

$$\boldsymbol{h}_t = \phi(\mathbf{W}_{xh}^{\top}\boldsymbol{x}_t + \mathbf{W}_{hh}^{\top}\boldsymbol{h}_{t-1} + \boldsymbol{b})$$

where $\boldsymbol{b}$ is the bias vector and $\phi(.)$ is the activation function (e.g., the ReLU or Tanh functions).

At the time step t, if we denote $\mathbf{W}_{hy}$ as the weight from the state $\boldsymbol{h}_t$ of the current recurrent neuron to the corresponding output denoted by $\boldsymbol{y}_t$, the output $\boldsymbol{y}_t$ is computed as follows:

$$\boldsymbol{y}_t = \phi(\mathbf{W}_{hy}^{\top}\boldsymbol{h}_t + \boldsymbol{c})$$

where $\boldsymbol{c}$ is the bias vector and $\phi(.)$ is the activation function (e.g., the softmax function).

7.3 Long short-term memory networks

Long short-term memory (LSTM) networks are a type of RNNs capable of learning long-term dependencies, first proposed by Hochreiter and Schmidhuber [Hochreiter and Schmidhuber(1997)] and gradually improved over the years by other works [Sak et al.(2014), Zaremba et al.(2014)]. An LSTM network was introduced to address the exploding and vanishing gradients problems as well as the short-term memory problem (i.e., the lost information from some of the first elements from the corresponding input in the memory cell of a long RNN) in training RNNs.

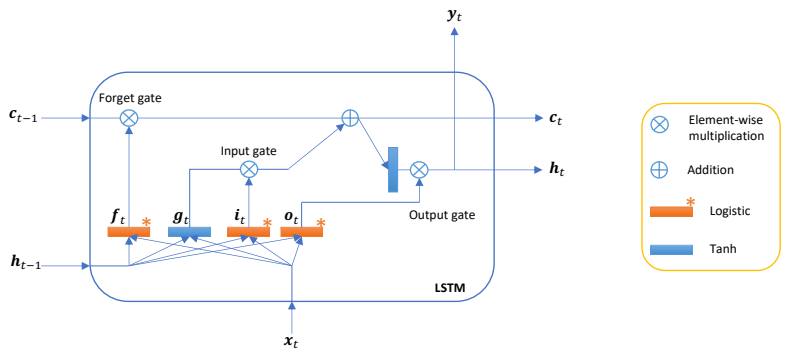


Fig. 11. An architecture of a long short-term memory (LSTM) network.

The key idea of an LSTM network is about storing long-term memory. An LSTM network can learn to figure out what information from the inputs should be read and stored in the long-term state denoted by $\boldsymbol{c}_t$ as well as what information should be thrown out from $\boldsymbol{c}_t$. A visualization of an LSTM network is shown in Figure 11. As depicted, there are four layers in an LSTM network including the main layer and three additional layers (i.e., gate controllers), namely, the forget gate, the input gate, and the output gate.

- The main layer at the time step t aims to analyse the current input vector $\boldsymbol{x}_t$ and the previous (short-term) state $\boldsymbol{h}_{t-1}$ to gain the output $\mathbf{g}_t$ (i.e., $\boldsymbol{h}_t$ in a basic cell RNN). In an LSTM cell, the layer's output $\mathbf{g}_t$ does not go straight out, but instead goes through a gate controller to decide what parts are stored in the long-term state (i.e., $\boldsymbol{c}_t$).
- Using the logistic activation function, the forget gate $\mathbf{f}_t$ aims to learn which parts of the long-term state $\boldsymbol{c}_t$ should be erased. The input gate $\mathbf{i}_t$ aims to control which parts of $\mathbf{g}_t$ should be added to the long-term state $\boldsymbol{c}_t$ while the output gate $\boldsymbol{o}_t$ aims to learn which parts of the long-term state $\boldsymbol{c}_t$ should be outputted for both $\boldsymbol{h}_t$ and $\boldsymbol{y}_t$.

The following equation (i.e., Eq. (8)) summarises the computing process of the four layers of an LSTM network at the time step t:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}_{xi}^\top \boldsymbol{x}_t + \mathbf{W}_{hi}^\top \boldsymbol{h}_{t-1} + \boldsymbol{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{W}_{xf}^\top \boldsymbol{x}_t + \mathbf{W}_{hf}^\top \boldsymbol{h}_{t-1} + \boldsymbol{b}_f\right) \\
\boldsymbol{o}_t &= \sigma\left(\mathbf{W}_{xo}^\top \boldsymbol{x}_t + \mathbf{W}_{ho}^\top \boldsymbol{h}_{t-1} + \boldsymbol{b}_o\right) \\
\mathbf{g}_t &= \tanh\left(\mathbf{W}_{xg}^\top \boldsymbol{x}_t + \mathbf{W}_{hg}^\top \boldsymbol{h}_{t-1} + \boldsymbol{b}_g\right) \\
\boldsymbol{c}_t &= \mathbf{f}_t \otimes \boldsymbol{c}_{t-1} + \mathbf{i}_t \otimes \mathbf{g}_t \\
\boldsymbol{y}_t &= \boldsymbol{h}_t = \boldsymbol{o}_t \otimes \tanh(\boldsymbol{c}_{t-1})
\end{aligned} \tag{8}$$

where $\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$ and $\mathbf{W}_{xg}$ are the weight matrices from the input vector $\boldsymbol{x}_t$ to each of the four layers while $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$ and $\mathbf{W}_{hg}$ are the weight matrices from the previous short-term state $\boldsymbol{h}_{t-1}$ to each of the four layers, and $\boldsymbol{b}_i$, $\boldsymbol{b}_f$, $\boldsymbol{b}_o$ and $\boldsymbol{b}_g$ are the bias vectors to each of the four layers, respectively. In general, the output $\boldsymbol{y}_t$ can be different from the short-term state $\boldsymbol{h}_t$ (i.e., $\boldsymbol{y}_t = \phi(\mathbf{W}_{hy}^\top \boldsymbol{h}_t + \boldsymbol{b}_y)$) where $\boldsymbol{b}_y$ is the bias vector while $\mathbf{W}_{hy}$ is the weight from $\boldsymbol{h}_t$ to $\boldsymbol{y}_t$, and $\phi(.)$ is the activation function (e.g., the softmax function).

8 CONCLUSION

In this paper, in addition to exploiting deep domain adaptation with automatic representation learning for SVD, we have successfully proposed a novel cross-domain kernel classifier leveraging the max-margin principle to significantly improve the capability of the transfer learning of software vulnerabilities from imbalanced labeled projects into imbalanced unlabeled ones in order to deal with two crucial issues in SVD including i) learning automatic representations to improve the predictive performance of SVD, and ii) coping with the scarcity of labeled vulnerabilities in projects that require the laborious labeling of code by experts. Our proposed cross-domain kernel classifier can not only effectively deal with the imbalanced datasets but also leverage the information of the unlabeled projects to further improve the classifier's performance. The experimental results show the superiority of our proposed method compared with other state-of-the-art baselines in terms of the representation learning and transfer learning processes.

ACKNOWLEDGEMENTS

Grundy is supported by ARC Laureate Fellowship FL190100035.

REFERENCES

[Abadi et al.(2016)] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.

[Ami et al.(2024)] Amit Seal Ami, Kevin Moran, Denys Poshyvanyk, and Adwait Nadkarni. 2024. "False negative – that one is going to kill you": Understanding Industry Perspectives of Static Analysis based Security Testing. *IEEE Symposium on Security and Privacy* (2024).

[Chawla et al.(2002)] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research* 16 (2002), 321–357.

[Chen et al.(2020)] Chao Chen, Zhihang Fu, Zhihong Chen, Sheng Jin, Zhaowei Cheng, Xinyu Jin, and Xian-Sheng Hua. 2020. HoMM: Higher-order Moment Matching for Unsupervised Domain Adaptation. *Thirty-Fourth AAAI Conference on Artificial Intelligence* (2020).

[Cheng et al.(2019)] Xiao Cheng, Haoyu Wang, Jiayi Hua, Miao Zhang, Guoai Xu, Li Yi, and Yulei Sui. 2019. Static Detection of Control-Flow-Related Vulnerabilities Using Graph Embedding. In *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*.

[Cortes and Vapnik(1995)] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning*.

[Cui et al.(2019)] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge J. Belongie. 2019. Class-Balanced Loss Based on Effective Number of Samples. *CoRR* abs/1901.05555 (2019).

[Dam et al.(2018)] Hoa K. Dam, Truyen Tran, Trang Pham, Shien W. Ng, John Grundy, and Aditya Ghose. 2018. Automatic feature learning for predicting vulnerable software components. *IEEE Transactions on Software Engineering* (2018).

[Dowd et al.(2006)] Mark Dowd, John McDonald, and Justin Schuh. 2006. *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Addison-Wesley Professional.

[Duan et al.(2019)] Xu Duan, Jingzheng Wu, Shouling Ji, Zhiqing Rui, Tianyue Luo, Mutian Yang, and Yanjun Wu. 2019. VulSniper: Focus Your Attention to Shoot Fine-Grained Vulnerabilities. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*.

[Duong et al.(2015)] Phuong Duong, Van Nguyen, Mi Dinh, Trung Le, Dat Tran, and Wanli Ma. 2015. Graph-based semi-supervised support vector data description for novelty detection. *International Joint Conference on Neural Networks (IJCNN)* (2015).

[Feng et al.(2020)] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *CoRR* abs/2002.08155 (2020).

[Fu et al.(2024a)] Michael Fu, Van Nguyen, Chakkrit Tantithamthavorn, Dinh Phung, and Trung Le. 2024a. Vision transformer inspired automated vulnerability repair. *ACM Transactions on Software Engineering and Methodology* 33, 3 (2024), 1–29.

[Fu et al.(2023a)] Michael Fu, Van Nguyen, Chakkrit Kla Tantithamthavorn, Trung Le, and Dinh Phung. 2023a. VulExplainer: A Transformer-based Hierarchical Distillation for Explaining Vulnerability Types. *IEEE Transactions on Software Engineering* (2023).

[Fu et al.(2024b)] Michael Fu, Jirat Pasuksmit, and Chakkrit Tantithamthavorn. 2024b. AI for DevSecOps: A Landscape and Future Opportunities. *arXiv preprint arXiv:2404.04839* (2024).

[Fu and Tantithamthavorn(2022)] Michael Fu and Chakkrit Tantithamthavorn. 2022. Linevul: A transformer-based line-level vulnerability prediction. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 608–620.

[Fu et al.(2024c)] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Yuki Kume, Van Nguyen, Dinh Phung, and John Grundy. 2024c. AIBugHunter: A Practical tool for predicting, classifying and repairing software vulnerabilities. *Empirical Software Engineering* 29, 1 (2024), 4.

[Fu et al.(2022)] Michael Fu, Chakkrit Tantithamthavorn, Trung Le, Van Nguyen, and Dinh Phung. 2022. VulRepair: a T5-based automated software vulnerability repair. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 935–947.

[Fu et al.(2023b)] Michael Fu, Chakkrit Kla Tantithamthavorn, Van Nguyen, and Trung Le. 2023b. Chatgpt for vulnerability detection, classification, and repair: How far are we?. In *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 632–636.

[Ganin and Lempitsky(2015)] Yaroslav Ganin and Victor Lempitsky. 2015. Unsupervised Domain Adaptation by Backpropagation. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37* (Lille, France) *(ICML'15)*. 1180–1189.

[Goodfellow et al.(2014)] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[Grieco et al.(2016)] Gustavo Grieco, Guillermo Luis Grinblat, Lucas Uzal, Sanjay Rawat, Josselin Feist, and Laurent Mounier. 2016. Toward Large-Scale Vulnerability Discovery Using Machine Learning. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (New Orleans, Louisiana, USA) *(CODASPY '16)*. 85–96.

[Hanif et al.(2021)] Hazim Hanif, Mohd H. Nizam, Mohd Faizal, Ahmad Firdaus, and Nor B. Anuar. 2021. The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *Journal of Network and Computer Applications* 179 (2021), 103009. https://doi.org/10.1016/j.jnca.2021.103009

[Hearst et al.(1998)] Marti A. Hearst, Susan T. Dumais, Edgar Osuna, John Platt, and Bernhard Schölkopf. 1998. Support vector machines. *IEEE Intelligent Systems and their Applications* 13, 4 (1998), 18–28. https://doi.org/10.1109/5254.708428

[Hochreiter and Schmidhuber(1997)] Sepp Hochreiter and Jűrgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[Hofmann et al.(2008)] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. 2008. Kernel methods in machine learning. *The Annals of Statistics*, (2008).

[Hsu and Lin(2002)] Chih-Wei Hsu and Chih-Jen Lin. 2002. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks* 13, 2 (2002), 415–425.

[Joachims(2006)] Thorsten Joachims. 2006. Training linear SVMs in linear time. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), 217–226.

[Kim et al.(2017)] Seulbae Kim, Seunghoon Woo, Heejo Lee, and Hakjoo Oh. 2017. VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 595–614.

[Kingma and Ba(2014)] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[Kipf and Welling(2016)] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* abs/1609.02907 (2016).

[Kotsiantis et al.(2006)] Sotiris B. Kotsiantis, Ioannis D. Zaharakis, and Panayiotis E. Pintelas. 2006. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review* 26 (2006), 159–190. https://api.semanticscholar.org/CorpusID:1721126

[Le et al.(2016)] Anh Le, Trung Le, Khanh Nguyen, Van Nguyen, Thai Hoang Le, and Dat Tran. 2016. Fast kernel-based method for anomaly detection. *International Joint Conference on Neural Networks (IJCNN)* (2016).

[Le et al.(2021)] Trung Le, Tuan Nguyen, Nhat Ho, Hung Bui, and Dinh Phung. 2021. LAMDA: Label Matching Deep Domain Adaptation. In *Proceedings of the 38th International Conference on Machine Learning*. 6043–6054.

[Le et al.(2015)] Trung Le, Van Nguyen, Anh Nguyen, and Khanh Nguyen. 2015. Adaptable linear support vector machine. *National Foundation for Science and Technology Development Conference on Information and Computer Science (NICS)* (2015).

[Le et al.(2014)] Trung Le, Van Nguyen, Thien Pham, Mi Dinh, and Thai Hoang Le. 2014. Fuzzy semi-supervised large margin one-class support vector machine. *The National Foundation for Science and Technology Development (NAFOSTED) Conference on Information and Computer Science* (2014).

[Le et al.(2010)] Trung Le, Dat Tran, Wanli Ma, and Dharmendra Sharma. 2010. An optimal sphere and two large margins approach for novelty detection. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*. 1–6.

[Li et al.(2016)] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Hanchao Qi, and Jie Hu. 2016. VulPecker: An Automated Vulnerability Detection System Based on Code Similarity Analysis. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications* (Los Angeles, California, USA) *(ACSAC '16)*. 201–213.

[Li et al.(2018a)] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, Zhaoxuan Chen, Sujuan Wang, and Jialai Wang. 2018a. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. *CoRR* abs/1807.06756 (2018).

[Li et al.(2018b)] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018b. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. *CoRR* abs/1801.01681 (2018).

[Lin and Wang(2002)] Chun-Fu Lin and Sheng-De Wang. 2002. Fuzzy support vector machines. *IEEE Transactions on Neural Networks*.

[Lin et al.(2020)] Guanjun Lin, Sheng Wen, Qing-Long Han, Jun Zhang, and Yang Xiang. 2020. Software Vulnerability Detection Using Deep Neural Networks: A Survey. *Proc. IEEE* 108, 10 (2020), 1825–1848. https://doi.org/10.1109/JPROC.2020.2993293

[Lin et al.(2018)] Guanjun Lin, Jun Zhang, Wei Luo, Lei Pan, Yang Xiang, Olivier De Vel, and Paul Montague. 2018. Cross-Project Transfer Representation Learning for Vulnerable Function Discovery. In *IEEE Transactions on Industrial Informatics*, Vol. 14.

[Lin et al.(2017)] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. *CoRR* abs/1708.02002 (2017).

[Liu et al.(2020)] Shigang Liu, Guanjun Lin, Lizhen Qu, Jun Zhang, Olivier De Vel, Paul Montague, and Yang Xiang. 2020. CD-VulD: Cross-Domain Vulnerability Discovery based on Deep Domain Adaptation. *IEEE Transactions on Dependable and Secure Computing* (2020). https://doi.org/10.1109/TDSC.2020.2984505

[Liu et al.(2023)] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D Le, and David Lo. 2023. Refining ChatGPT-generated code: Characterizing and mitigating code quality issues. *ACM Transactions on Software Engineering and Methodology* (2023).

[Liu et al.(2022)] Yue Liu, Chakkrit Tantithamthavorn, Yonghui Liu, Patanamon Thongtanunam, and Li Li. 2022. Autoupdate: Automatically recommend code updates for android apps. *arXiv preprint arXiv:2209.07048* (2022).

[Long et al.(2015)] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. 2015. Learning Transferable Features with Deep Adaptation Networks. In *Proceedings of the 32nd International Conference on Machine Learning*. 97–105.

[Maaten and Hinton(2008)] Laurens V. D. Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* (2008).

[Menon et al.(2021)] Aditya Krishna Menon, Sadeep Jayasumana, Ankit Singh Rawat, Himanshu Jain, Andreas Veit, and Sanjiv Kumar. 2021. Long-tail learning via logit adjustment. *International Conference on Learning Representations* (2021).

[Neuhaus et al.(2007)] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. 2007. Predicting Vulnerable Software Components. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. 529–540.

[Nguyen et al.(2017)] Tu D. Nguyen, Trung Le, Hung Vu, and Dinh Phung. 2017. Dual Discriminator Generative Adversarial Nets. *CoRR* abs/1709.03831 (2017).

[Nguyen et al.(2020)] Van Nguyen, Trung Le, Olivier De Vel, Paul Montague, John Grundy, and Dinh Phung. 2020. Dual-Component Deep Domain Adaptation: A New Approach for Cross Project Software Vulnerability Detection. *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2020).

[Nguyen et al.(2019)] Van Nguyen, Trung Le, Tue Le, Khanh Nguyen, Olivier DeVel, Paul Montague, Lizhen Qu, and Dinh Phung. 2019. Deep Domain Adaptation for Vulnerable Code Function Identification. In *The International Joint Conference on Neural Networks (IJCNN)*.

[Nguyen et al.(2014)] Van Nguyen, Trung Le, Thien Pham, Mi Dinh, and Thai Hoang Le. 2014. Kernel-based semi-supervised learning for novelty detection. *International Joint Conference on Neural Networks (IJCNN)* (2014).

[Nguyen et al.(2022a)] Van Nguyen, Trung Le, Chakkrit Tantithamthavorn, John Grundy, Hung Nguyen, Seyit Camtepe, Paul Quirk, and Dinh Phung. 2022a. An Information-Theoretic and Contrastive Learning-based Approach for Identifying Code Statements Causing Software Vulnerability. *CoRR* abs/2209.10414 (2022).

[Nguyen et al.(2021)] Van Nguyen, Trung Le, Olivier De Vel, Paul Montague, John Grundy, and Dinh Phung. 2021. Information-theoretic source code vulnerability highlighting. *International Joint Conference on Neural Networks (IJCNN)* (2021).

[Nguyen et al.(2022b)] Van-Anh Nguyen, Dai Quoc Nguyen, Van Nguyen, Trung Le, Quan Hung Tran, and Dinh Q. Phung. 2022b. ReGVD: Revisiting Graph Neural Networks for Vulnerability Detection. *International Conference on Software Engineering (ICSE)* (2022).

[Pornprasit and Tantithamthavorn(2021)] Chanathip Pornprasit and Chakkrit Kla Tantithamthavorn. 2021. JITLine: A simpler, better, faster, finer-grained just-in-time defect prediction. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 369–379.

[Pornprasit and Tantithamthavorn(2022)] Chanathip Pornprasit and Chakkrit Kla Tantithamthavorn. 2022. Deeplinedp: Towards a deep learning approach for line-level defect prediction. *IEEE Transactions on Software Engineering* 49, 1 (2022), 84–98.

[Rahimi and Recht(2008)] Ali Rahimi and Benjamin Recht. 2008. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems*.

[Rumelhart et al.(1986)] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning Internal Representations By Error Backpropagation. In *Parallel Distributed Processing*, Vol. 1.

[Russell et al.(2018)] Rebecca L. Russell, Louis Y. Kim, Lei H. Hamilton, Tomo Lazovich, Jacob A. Harer, Onur Ozdemir, Paul M. Ellingwood, and Marc W. McConley. 2018. Automated Vulnerability Detection in Source Code Using Deep Representation Learning. *CoRR* abs/1807.04320 (2018).

[Sak et al.(2014)] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. 2014. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *CoRR* abs/1402.1128 (2014).

[Schölkopf et al.(2001)] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. 2001. Estimating the Support of a High-Dimensional Distribution. *Neural Comput.* 13, 7 (July 2001), 1443–1471.

[Schölkopf and Smola(2002)] Bernhard Schölkopf and Alexander J. Smola. 2002. Learning with Kernels. *The MIT Press* (2002).

[Schölkopf et al.(2000)] Bernhard Schölkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. 2000. Neural Computation. *Machine Learning*.

[Shin et al.(2011)] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A. Osborne. 2011. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Transactions on Software Engineering* 37, 6 (2011), 772–787.

[Shu et al.(2018)] Rui Shu, Hung H. Bui, Hirokazu Narui, and Stefano Ermon. 2018. A DIRT-T Approach to Unsupervised Domain Adaptation. In *International Conference on Learning Representations*.

[Sutskever et al.(2014)] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[Tax and Duin(2004)] David M.J. Tax and Robert P.W. Duin. 2004. Support Vector Data Description. *Journal of Machine Learning Research* 54, 1 (2004), 45–66.

[Thongtanunam et al.(2022)] Patanamon Thongtanunam, Chanathip Pornprasit, and Chakkrit Tantithamthavorn. 2022. Autotransform: Automated code transformation to support modern code review process. In *Proceedings of the 44th international conference on software engineering*. 237–248.

[Tsang et al.(2007)] Ivor W. Tsang, Andras Kocsor, and James T. Kwok. 2007. Simpler Core Vector Machines with Enclosing Balls. In *Proceedings of the 24th International Conference on Machine Learning (ICML '07)*. 911–918.

[Tsang et al.(2005)] Ivor W. Tsang, James T. Kwok, Pak-Ming Cheung, and Nello Cristianini. 2005. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research* 6 (2005), 363–392.

[Vapnik and Lerner(1963)] Vladimir N. Vapnik and Alexander Y. Lerner. 1963. Pattern recognition using generalized portrait method. *Automation and Remote Control* 24.

[Wattanakriengkrai et al.(2020)] Supatsara Wattanakriengkrai, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Hideaki Hata, and Kenichi Matsumoto. 2020. Predicting defective lines using a model-agnostic technique. *IEEE Transactions on Software Engineering* 48, 5 (2020), 1480–1496.

[Weston et al.(2003)] Jason Weston, Bernhard Schölkopf, and Gökhan Bakir. 2003. Learning to Find Pre-Images. *Advances in Neural Information Processing Systems* 16.

[Yamaguchi et al.(2011)] Fabian Yamaguchi, Felix Lindner, and Konrad Rieck. 2011. Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning. In *Proceedings of the 5th USENIX conference on Offensive technologies*. 13–23.

[Zaremba et al.(2014)] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent Neural Network Regularization. *CoRR* abs/1409.2329 (2014).

[Zhuang et al.(2020)] Yuan Zhuang, Zhenguang Liu, Peng Qian, Qi Liu, Xiang Wang, and Qinming He. 2020. Smart Contract Vulnerability Detection using Graph Neural Network. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20.*

[Zimmermann et al.(2009)] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. 2009. Cross-project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering* (Amsterdam, The Netherlands) *(ESEC/FSE '09)*. 91–100.